Ontology Mapping for Dynamic Service Invocation On the Semantic Web

Mark H. Burstein
BBN Technologies

Abstract

The need for translation during the dynamic invocation of services described on the semantic web is not addressed merely by introducing middle agents to translate messages. When an agent is attempting to utilize a service discovered and described on the Semantic Web using a service language like OWL-S, it must be able to translate descriptions of required inputs, given in terms from the ontology used in the service description, into corresponding terms from a local ontology *in order to formulate a proper request*. We are currently developing a model of the service invocation process that folds in translation. We are also addressing the issue raised here as we develop an abstract architecture for semantic web services, our charter on the Architecture Committee of the Semantic Web Services Initiative (SWSI).

Introduction

A key objective for semantic web services is to provide machine interpretable descriptions of web services so that other software agents can use them without having any prior 'built-in' knowledge about how to invoke them, just as people can make effective use of web sites found using a search engine. Unless we can guarantee that the services discovered utilize exactly the same ontologies as the potential client, this kind of increased interoperability hinges on an ability to interpret descriptions that were developed independently, using at least some different ontologies than those used by the client for its own local reasoning.

I have argued in [1] that the 'traditional' approach to inserting translation support into this kind of architecture, interposing a middle agent between client C and server S to translate messages from language L_C into language L_S, is going to prove inadequate. The reason for this hinges on the fact that the message that C needs to send must be constructed dynamically by reasoning about the relationship between its own goals and the invocation message content requirements provided in the service description. I briefly review the argument here, by example, and indicate its implications for our current work.

Service Invocation Reasoning Across Ontologies

The model of semantic web services we have used in the DAML-S coalition as the basis for our design of the DAML-S [3] and now OWL-S ontology is based, in large part, on the idea that such services that can be dynamically discovered, applied and composed by reasoning from published representations of their *service descriptions,* expressed in a declarative semantic web description language like OWL [5]. Once service descriptions are published, other software systems (agents) can reason about how to invoke the services dynamically by reading these descriptions at run time, composing request messages with the prescribed content, and sending them via a transport model such as WSDL [6]

This technology should ultimately enable individuals to use their own personal software agents for such things as comparative shopping, information discovery, or to weave together new services from available ones, much as a travel agent does. Ultimately, though, this kind of interoperability means having an ability to reach out to communities with services that were developed independently, using different ontologies. When the software agent that is going to invoke a service is developed by one organization, using internal data models described in terms of one ontology, and the just-discovered service to be invoked is developed by another organization using a different ontology, the ontologies may use different terms for the same or similar notions. In such cases, translation is required, even though both software systems seek to support interoperability by using DAML or OWL ontologies to describe themselves and the data they can reason about. To make ontologies interoperable, *ontology mappings* are required, so that the terms in the different ontologies are brought into correspondence. Sometimes these mappings can only be partial, as ontologies developed for different purposes may not have needed to define all of the same concepts, or may have incompatible definitions. The descriptions of such mappings are sometimes called *articulations* [8], or *bridging axioms* [11] and can include simple correspondences between terms (uni or bi-directional), rules 'defining' terms in one ontology in terms of some set of terms in another, and even functional mappings, such as for translating units of measure.

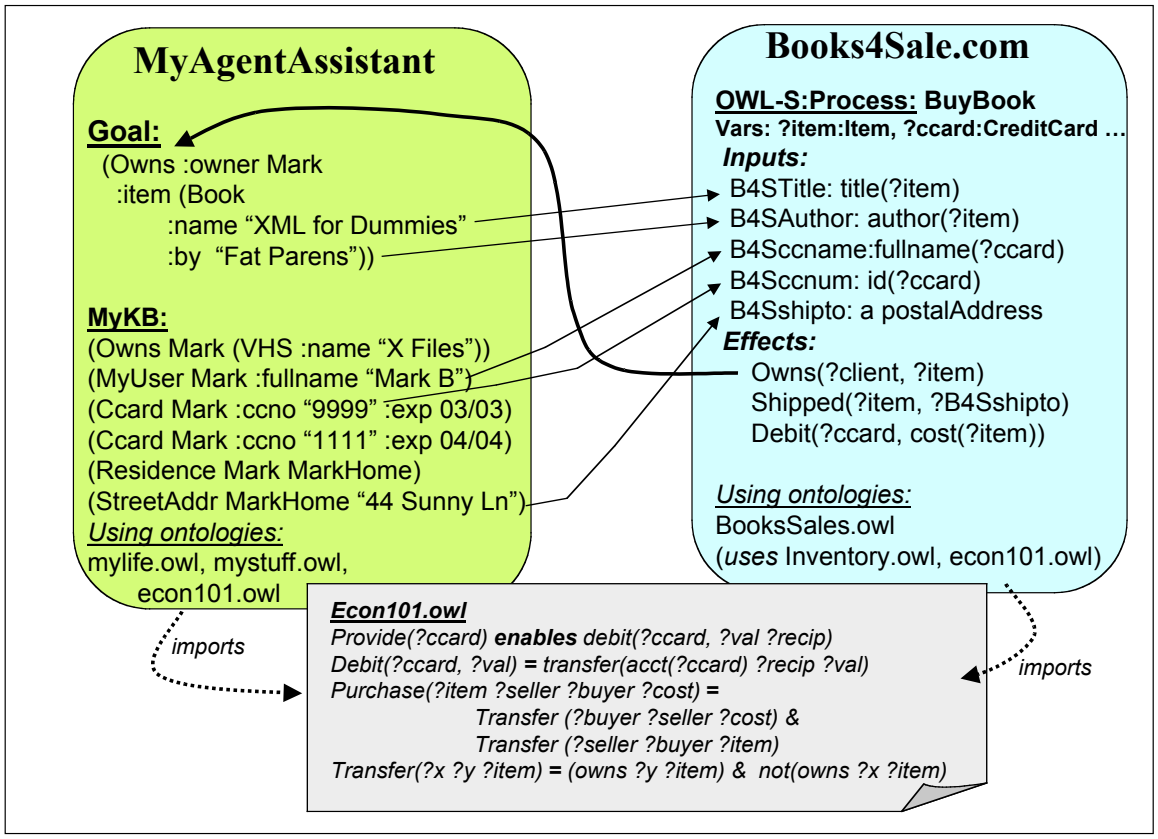

Figure 1: Book Buying Example

Now consider the example in Figure 1. In this hypothetical example, I use my personal agent to help me buy a book about XML on the web. My agent knows a number of things about me and my possessions, and it knows how to shop for me by contacting commercial semantic web services. MyAgent uses a matchmaker to find a reputable service called Books4Sale, which has published a process model using OWL-S declaring that if I invoke this service successfully, the service *outcome* will be that I own the book requested, provided that it is in its inventory. The process description specifies as 'inputs' the book title, author, a credit card number, and shipping information, described in the ontology used by that service. It produces as output (if successful) an order confirmation number and shipping tracking number (not shown).

At first blush, it would seem that MyAgent can now achieve my goal by translating its representation of that goal (internally described using its own local ontologies **mylife** and **mystuff**) into the form described by the inputs to the Books4Sale's process description. In fact, however, my goal, owning a book, corresponds not to the description of the process and its inputs, but to a stated *effect* of the service process description, that I own whatever book was requested when the process has completed successfully.

A classical AI planning system would rely on just such a correspondence between goals and operator effects, and MyAgent could plan to reduce its goal to an action, by treating this service description essentially as a *planning operator* and using it to build a plan that treats the Books4Sale process as a step initiated by message passing. It would then execute the plan by sending Books4Sale a message to perform the specified process that included the process' required inputs, a message described for the service separately in WSDL. The mapping or relationship between the abstract process input descriptions and the fields of the WSLD message are defined in the OWL-S *grounding* for that process. OWL-S was designed to support just this kind of reasoning by providing an ontology for representing in OWL a standard way to associate inputs, outputs, preconditions and effects (IOPEs) with represented processes, so that potential service clients can treat service processes as *descriptions of planning operators* ([10,13] ,among others, demonstrate this), and also grounding relations between these descriptions and WSDL. MyAgent can thus reason that by invoking the BuyABook process with a message containing the required attributes of the desired book (title, author) specified among the other required process inputs. The process can be expected to succeed because the process description

says that the item owned after the process completes will have those attributes of the book in my goal.

Translation During Request Formulation Reasoning

It is at this point in the agent's reasoning that translation can be required. In our example, we show (Figure 1) the BuyABook process is described using OWL-S by Books4Sale.com as requiring an input called B4S*title* whose value is constrained to be the *title* of the item to be purchased (owned) – as referenced also in the *effect* part of the process description. To fill this input requirement, my agent needs to reason deductively using its own knowledge base to identify the value of a some property of books that *translates or maps to* the *title* property in the service's ontology associated with the class *item* representing its inventory. Say that MyAgent uses an ontology with properties *name, by, pubdate, purchdate…* for items in the class *book* in its collection. To do this translation reasoning, it needs a set of *bridging axioms* [11] that define the relationships between terms in this ontology and terms in the ontology used by Books4Sale where *title* is a property of elements of the class *item* representing inventory items. Assuming that this articulation exists as shown in Table 1 in a published mapping relating the two ontologies (which may well be used by others on the semantic web as well), then the agent can determine that it needs to find values of the *name* property for the book sought and provide that string as the value as the B4S*title* property in its OWL-S description of the process step being constructed.

MyStuff	*B ridging relation*	**Inventory**
Book	*InstClass-SetClass*	Item
Book.name	*equivInstanceProp*	Item.title
Book.by	*equivInstanceProp*	Item.author
Book.pubDate	*equivInstancerProp*	Item.PubDate
Book.purchDate	- N/A -	
Book.shelfLocatio n	- N/A -	
	- N/A -	Item.qty

Table 1: Mappings between MyStuff.owl and Books4Sale.owl

Table 1 shows that the relationships between the two ontologies is not simply one of term equivalence. In the MyStuff ontology, elements of the class Book are specific physical objects sitting on particular shelves, where in the Inventory model used by Books4Sale, each Item description represents a quantity of like items in the stock of that company. Nonetheless, these two concepts can be related as some of the descriptive properties of the class Book in the MyStuff ontology are equivalent to descriptive properties of Items in the Inventory ontology used by Books4Sale. For example, each instance of type Item represents of some number of books all of whcih of have the same title, author, firstPubDate, etc. For this reason, we can utilize bridging axioms that capture the conditions under which an Inventory:Item description (with qty =1) can refer to the same entity as a description of a MyStuff:Book.

Thus far, we have shown how several of the required inputs to the BuyBook process could be identified by reasoning about the relationship between the client agent's goal, and the effects of a published process description of an unfamiliar service provider. The identification of these inputs hinges on two things:

1. The client can match its goal(s) to (a translation of) some of the proposed process' effect(s), unifying items referenced as arguments to these goals with the variables in those effects, and .
2. The type restrictions and preconditions specified for input variables referenced in these effects translate to restrictions in the client's ontology consistent with the objects specified in the client's goals.

In short, as in classical planning, the process' effects must unify with the client's goal, and the process conditions must be satisfied, but the unification process of the planner can only succeed if these constraints can be translated into the client's native ontology. In the end it is not the input parameters that must be translated – these are variables. What must be translated are the effects and constraints that reference these inputs. And the translation of these elements is from the ontology of the service provider into the ontology of the client where the planning takes place.

Identifying values for non-goal related inputs

Unfortunately, even if successful, the reasoning just described is not generally enough to determine all of the inputs to a service. There is also the question of providing the required inputs that are not part of explicit goals of the client. To illustrate these, we will talk about the credit card and shipping information inputs in our sample book buying process.

Oftentimes, processes will require inputs that do not directly reflect elements of clients goals, but can nonetheless be determined either directly based on knowledge that the client has, or as a result of additional decisions made by the client. We will use the requirement for shipping address and credit card information respectively to illustrate these to cases. In fact, either one could require the client make a decision that was not anticipated in advance.

One effect of the BuyBook process is that, when successful, a book is shipped to the address specified by input B4Sshipto. For the purposes of this example, we will assume that MyAgent was designed to handle this using a general rule that all purchases should be shipped to the address of the buyer (user). (Obviously there could be exceptions, such as when the purchase was a gift.)

Another effect of the process is that a credit card is debited by the amount of the purchase (ignoring taxes). Indeed, the process should also have a precondition that the credit card identified by the inputs has sufficient credit available. For our example, let's assume that MyAgent knows I, the user, have three credit cards with known levels of available credit, one of which I use only for business purposes. A decision that MyAgent needs to make in order to invoke the BuyBook process is *which credit card should be used for the purchase*. This decision can be made based on a combination of constraints specified by the service provider in the published process description (e.g., the amount of available credit constraint) and internal constraints or preferences specified by the user (business use vs. personal use, prefer the card with the lowest interest rate etc.) As a fallback, MyAgent should be able to ask for the user to help with such decisions.

Consider the shipping address first. This BuyBook process input is associated with an effect that the Item referred to (by B4Stitle and B4Sauthor) is sent to the B4Sshipto address when the process completes successfully. Let's assume for a moment that this effect was not specified explicitly as a goal by the user. MyAgent must still determine a value for that process input for the process to succeed. When such effects are not explicit goals, but are associated with required inputs, then those inputs must be treated like informational preconditions that must be established to use the process successfully. Oftentimes, default policies or rules can be used to cover such cases. A default rule here might be something like the following: *"Unless otherwise specified by the user, purchases by MyAgent for user U should be sent to U's home."* This rule would apply to all purchases made using semantic web services when those services ask for a mailing address.

The translation issue here is that a policy of this kind can only be applied if MyAgent can identify the mapping of the effect (*shipped* ?item ?addr) in Books4Sale's ontology and to a corresponding description in a default rule or policy specified for MyAgent. This again argues that the effects and other constraints on the process' input variables must be translated back into the ontology of the client, so that the client can determine permissible input values. After these values have been identified, Further translation reasoning may be required to coerce the values into an appropriate form, such as converting a *location* to an *address* in the proper format.

Our second example reinforces this point. and addresses the possible need for constraint translation into the client's ontology to support dynamic decisionmaking with regard to process inputs. Figure 1 shows that the BuyBook process requires as input a credit card number and expiration date to complete the purchase. This credit card is debited, an effect of the process which is clearly not a goal of the user, but is a means to that end, recognized as part of a broadly shared economic transaction model associated with a shared ontology. As Figure 1 also shows, MyAgent could be aware of several credit cards belonging to the user. The client needs to decide which input values (here, identifying a credit card) are acceptable (so pick one) or preferred while also satisfying the constraints identified as input or preconditions (know the number and expiration date, sufficient credit available on the card).

As with the shipping address, the translation issue is that the agent must first recognize what decision is required. This decision corresponds to the selection of a binding for the variable identifying a credit card during planning. Allowed values satisfy all of the constraints explicitly identified as preconditions and input type constraints. Note here that a critical constraint is that the credit card number and expiration date are for the same card. These constraints must first be translated into the client's ontology. The client can then use internal policies to select among possible candidate credit cards using whatever policies the agent had for use of those entities in its own knowledge base.

Once a card is selected, the client merely needs to translate the values of the attributes of the selected credit card (id number and expiration date) that are required as inputs in the message sent to the service. (Note that there is no guarantee that these attributes have the same names in both ontologies, but that this was addressed by the reverse translation of the input constraints.) In this case, both are normally strings, which normally require no further translation. However, it could have been the case that MyAgent stored the expiration date as two numbers, a separate month and year.

Implementation Approach

We (Burstein & McDermott) are currently in the process of implementing the approach to interleaved planning and translation for semantic web services sketched above using the estimated-regression planner, Unpop [9,10] developed at Yale University. This joint work also extends our approach to translation for web service composition planning sketched in [11], and the model of Ontology translation described in [7]. The approach is based on first merging ontologies while keeping their terms distinct, and then introducing bridging axioms to relate the terms. The semantic web allows ontologies to be combined and interrelated in this way by using namespaces to distinguish the terms of different ontologies using URIs. We make the additional assumption that for interoperability between inter-translatable ontologies, bridging axioms defining correspondences between terms in different ontologies must also be openly shared on the web. This paper is assumes that these bridging axioms already exist and are available to the client reading the service models of unfamiliar services, so that the question becomes one of ensuring that the mappings, even if they can only defined partially, are sufficient how to be used for the task at hand. Clearly, there will be times when these bridging rules are incomplete or absent, and a human must intervene to define the correspondences before the agent can reason with them. This is not addressed here.

As the examples in the previous section suggest, translation is required at several stages in the process of using a service description for service composition or request invocation. Assuming for the moment that the service to be used has been found by a discovery process, such as use of an OWL-S Matchmaker [12]. Then the client must do all of the following:

1. Read the service process model, loading all ontologies referenced within that model.
2. Find and load all bridging axioms defining relationships between the ontologies used by the service description provider and those used for the client's own internal reasoning and planning.
3. Translate all process effects and (precondition or type) constraints on input variables reflected in the service model in terms of its own ontologies.
4. Unify the effects of the process with the current planning goals, as expressed in its local ontologies. This is a normal step in the planning.
5. Select bindings for unbound input variables that satisfy all of the operator preconditions, and type constraints, and consistent with the effect bindings.
6. Translate the descriptions bound as input variables into the ontology of the service process model.
7. Apply the process grounding to map the inputs into the format of the request message, then send it.

Steps 1, 4, 5 and 7 are those that have been implemented in systems that use AI planners to compose and execute DAML-S or OWL-S service process models. Steps 2, 3 and 6 are the new steps required to do this when the ontology used by the client differs from those of the service providers.

Steps 2 and 3 together describe the problem of using pre-existing ontology mapping rules (bridging axioms) to translate an OWL-S process model into the client's ontologies. Some ontologies are assumed to be shared (such as OWL-S itself), requiring no translation, while others may have been developed independently. The critical elements to be translated are the domain-specific elements that are implicated in reasoning by the planner: the types of objects (item -> book, for example), and the relations used to identify or constrain the use of objects of those types in processes.

As described in [7], we treat the translation process as an application of bridging axioms with either a forward or backward chaining first order logical theorem prover. The result is a collected set of projections of terms from one ontology into terms of a target particular (set of) ontologies. The critical test of the success of such projections for this application is that all of the constraints represented within a service process model have been successfully translated, so that the critical planning decisions can be performed properly.

Step 6 amounts to translation of the request into a form that enables it to be communicated. Once the planner has proved that the process model will achieve the desired effect given an identified set of inputs represented in the client's ontology, it must formulate descriptions of those inputs in the service provider's ontology prior to grounding the process in WSDL. When the inputs are literals, this process is essentially trivial, but it need not be so. Suppose, for example, the BuyBook process required a "deliver by" date, selected by the client within some range constraints identified in the process model. Suppose further that the client represented dates as complex objects with a month, day and year, while the service required a fixed format string. This transformation would have to be performed to instantiate the operator, effectively by providing a complete set of inputs in service process model's native ontology so that the grounding could be applied in order to transmit the request.

Our contention here is that process request message formulation based on published process models requires tightly coupled support for ontology translation during planning, in order to translate the constraints on operator input variables. The client cannot simply formulate a message based on its goal and then have it translated since it must interpret the service model in order to do the plan reasoning required to establish the necessary elements of the request message. Thus, request formulation for published service process models is most easily provided using a planner that can reason directly with bridging axioms when establishing operator preconditions and variable value assignments for the operators represented by those process models. In such cases, the translation is implicitly performed directly by the client by utilizing the bridging axioms of the merged ontologies during the inferential reasoning associated with precondition establishment and variable value selection.

Translating Responses to Service Requests

Of course, sending a request to a server is only half the battle. The response must also be interpreted, and this too requires dynamic translation, of a kind for which OntoMerge and similar approaches to ontology translation are well suited. Here, the information to be provided is available as a coherent description to the service provider, and the issue is one of identifying a mapping of this abstract message description into the recipient's ontology. Most of the work on heterogeneous information retrieval is focused on the problem of developing the necessary ontology mappings to support this kind of translation.

In most cases, response translation can be done by any agent that has access to the source and target ontologies and the necessary articulation rules. This could be the sender, receiver or a middle agent. In previous work [2] we described an approach to message translation between agents based on the idea of generating special purpose translation code that would translate specific classes of structured data sets between representations used by

different agents. The approach assumed that the ontologies of both agents were available, along with axioms that related the two and a target data pattern. For example, if agent A was to answer a query Q by agent B, but the two used different ontologies, then our translation middle agent would develop a specific translator for the class of data sets $\mathbf{D_A}$ represented using ontology $\mathbf{O_A}$ that resulted from queries like Q, mapping them into data sets of class $\mathbf{D_B}$, represented in the target ontology $\mathbf{O_B}$. The code that was generated mapped a particular schema in one ontology into a different schema represented in a second ontology. This special purpose piece of code could then be stored in a middle agent, or provided to either the sender or receiver to make that particular class of messages translate efficiently in the future communications between those agents.

The difficulty with this approach for interoperation of web services is that it relies on being provided a specific target representation for the data to be translated, rather than an ontology. In contrast, services or agents may generate very different responses to a request, depending on their internal state. For example, a request to purchase something may result in a confirmation message or an "out of stock" message. Or a query might result in a response consisting of a heterogeneous list of object descriptions. Generating translations of such variable responses would require a large number of such stored procedures, and so the more recent work described in [7] has used a first order reasoning system to perform more dynamic message translation. Somewhat similar techniques have also been used in [4,8], although the latter uses a more syntactic kind of transformation rule.

Conclusion

We have argued by example for an approach to request formulation when interacting with dynamically discovered semantic web services that takes account of the need for ontology translation during the reasoning required to plan for these service requests. We claim that examples like this show why, as a practical matter, service requests cannot formed and then translated, but should be created by the service requester by *plan formation reasoning interleaved with translation by articulation rules*. Effectively, the planning process must decide which process to invoke (based on effects) and then plan (generate) the message by gathering the inputs required, translating constraints on what those inputs are into its own ontology along the way.

We are currently implementing this approach to service invocation with translation support in our joint work with Drew McDermott's group at Yale University.

Acknowledgements

I wish to thank my colleagues Drew McDermott of Yale University and the members of the DAML-S consortium for discussions on related topics. This writing of this paper was funded under a contract with Yale University to the DARPA DAML program, contract number F30602-00-0600.

References

[1] Burstein, Mark "The Many Faces of Mapping and Translation for Semantic Web Services" To appear in *Proceedings of WISE-03*, December 2003.

[2] Burstein, M., McDermott, D., Smith, D. and Westfold, S. "Derivation of glue code for agent interoperation." *J. Autonomous Agents and Multi-Agent Systems,*6:265-286. 2003.

[3] DAML Services Coalition (alphabetically A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), "DAML-S: Semantic Markup for Web Services", in *Proceedings of the International Semantic Web Working Symposium (SWWS)*, July 30-August 1, 2001.

[4] Chalupsky, H., "OntoMorph: A translation system for symbolic knowledge," In *Principles of Knowledge Representation and Reasoning: Proceedings of the Seventh International Conference (KR2000),* A. G. Cohn, F. Giunchiglia, and B. Selman, Editors. 2000, Morgan Kaufmann Publishers: San Francisco, CA.

[5] 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/owl-features/, 2003.

[6] Christensen, E. Cubera, F. Meredith, G. and Weerawarana, S. "Web Services Description Language (WSDL)" http://www.w3.org/TR/owl-ref/ 2002

[7] Dejing Dou, Drew McDermott, and Peishen Qi Ontology translation by ontology merging and automated reasoning. In *Proc. EKAW Workshop on Ontologies for Multi-Agent Systems*, 2002.

[8] Mitra, P., G. Wiederhold, and M. Kersten. "A Graph-Oriented Model for Articulation of Ontology Interdependencies" In *Proc. Conference on Extending Database Technology 2000* (EDBT'2000). 2000. Konstanz, Germany

[9] McDermott, D. "A Heuristic Estimator for Means-ends Analysis in Planning" In *Proceedings of the International Conference on AI Planning Systems,* pp. 142-149. 1996.

[10] McDermott, D. Estimated-Regression Planning for Interactions with Web Services. In *Proceedings of AI Planning Systems Conference*, 2002

[11] McDermott, D. Burstein, M. and Smith, D. "Overcoming ontology mismatches in transactions with self-describing agents." In *The Emerging Semantic Web: Selected Papers from the First Semantic Web Working Symposium*, pp.228-244. 2002.

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

[13] Wu, Dan, Parsia, Bijan, Sirin, Evren, Hendler, James, and Nau, Dana. Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.