

Dynamic Invocation of Semantic Web Services That Use Unfamiliar Ontologies

Mark H. Burstein, *BBN Technologies*

The Semantic Web research community has developed ontologies and languages for semantically describing Web Services. For example, OWL-S (www.daml.org/services/owl-s) is an ontology developed using the OWL Semantic Web description language.¹ The purpose of OWL-S, formerly DAML-S,² is to describe Web Services so that

software agents can read these descriptions and reason about how to interact with the services they describe. These efforts aim to facilitate interactions with services when the client agents lack built-in code for invoking those services' APIs—just as people can effectively use Web sites they've found through search engines. For example, without any reprogramming, a software system could have the flexibility to use various services that do the same kind of job but have different APIs. This can be useful in a number of situations, such as when a familiar service's API has changed or when the API is unavailable but a discovered new service could do the job. These semantic service descriptions also make it easier to *compose services*, by enabling reasoning about how to transform one service's outputs into another's inputs.

Semantic descriptions of services can't solve the interoperability problem by themselves, even if they all use the same semantic representation language. Increased interoperability also hinges on the ontologies used to represent the provided services' domains, including the terms used for the information required to request the services, their conditions of use, outputs, and effects. When communities develop services independently, they tend to use different ontologies to describe those services. Thus, interoperability also requires semantic translation between the representations clients use to formulate service goals and the representations the services use to accept requests and return results.

The traditional approach to inserting this kind of translation support into a distributed, multiagent

architecture is to insert middle agents between each client and server to translate the messages sent between them, effectively hiding the translation process from the agents doing the work. This method is inadequate when dynamically invoking unfamiliar services, because much of the translation reasoning that must happen occurs when the client is gathering the necessary inputs to invoke the service, well before a message has been sent. The client must construct its request message dynamically by reasoning about the relationship between its own goals and background knowledge and the information required to formulate a proper request message, as specified in the service's published description. Because that description might be written using a nonnative (to the client) ontology, it too must be translated. In this article, I present arguments why the translation task might best be done by the client rather than a by middle agent.

Service invocation reasoning across ontologies

The DAML-S Coalition developed OWL-S around the idea that the Semantic Web will provide many domain ontologies in a declarative Semantic Web description language such as OWL.¹ By using these ontologies in conjunction with a widely shared ontology for describing the structure of services themselves, we can promote the dynamic interoperability missing with current generation Web Service description languages such as WSDL.³ Semantic service descriptions include not only the data types of input and output message fields but also the seman-

The Semantic Web allows different ontologies to be used for describing different Semantic Web Services. Invoking an unfamiliar service can require several types of translation each of which may be done by different agents.

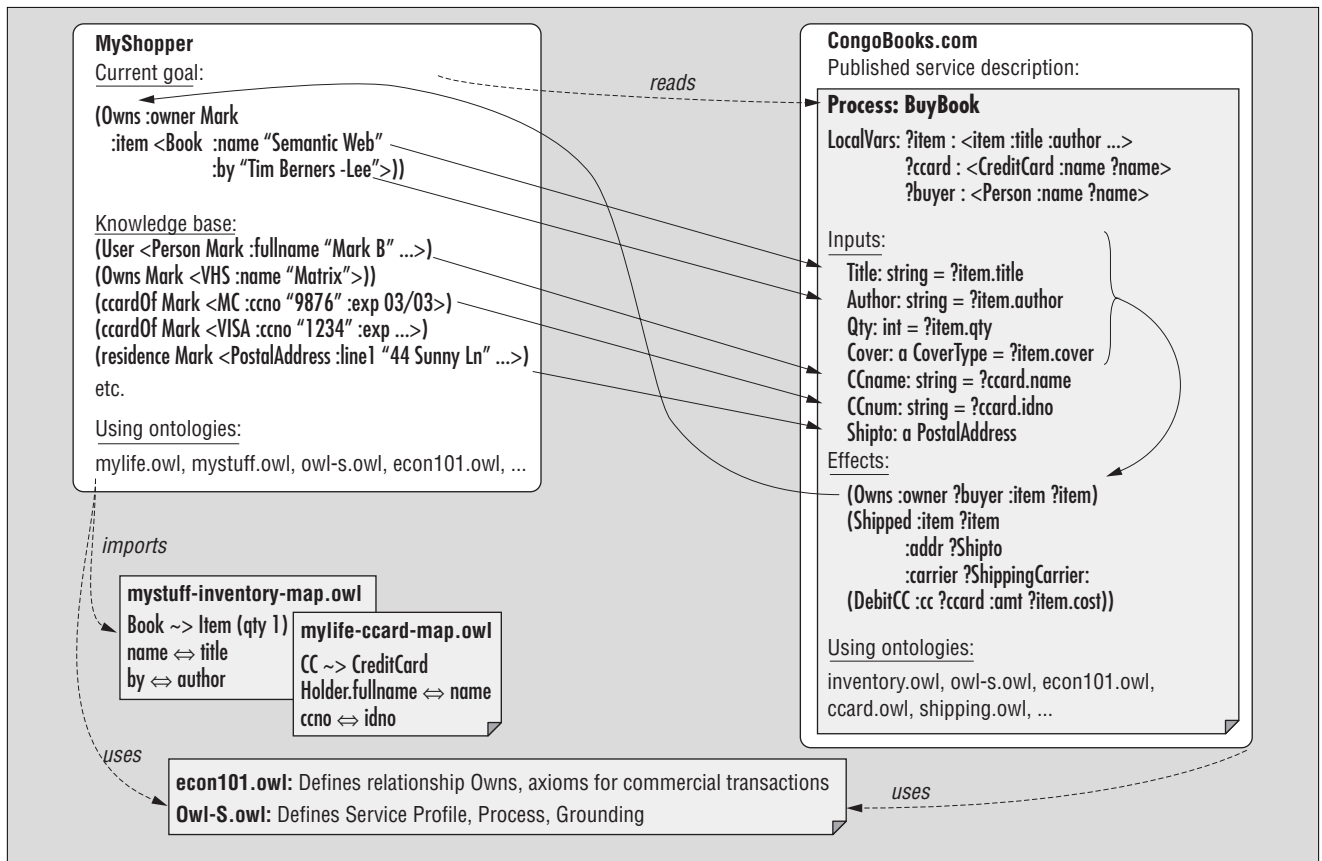


Figure 1: Book buying example.

tic categories of the objects those inputs refer to and their relationship to the service's effects. Indeed, OWL-S has now been used to invoke Web Services like those available at Amazon.com, using descriptions interpreted by the running client, which composes WSDL request messages with the prescribed content.⁴

This technology should enable individuals to use their personal software agents for such tasks as comparative shopping, information discovery, or weaving together available services into new ones, much as a travel agent does. Ultimately, though, this kind of interoperability is most useful when interacting with communities who developed services independently with different ontologies. When the software agent invoking a service is developed by one organization using internal data models described using one set of ontologies, and the discovered service is developed by another organization using a different set of ontologies, the ontologies might use different terms for the same or similar notions. Such cases require translation, even though both software systems seek to support interoperability by using

OWL ontologies to describe themselves and the data they can reason about. Making ontologies interoperable requires *ontology mappings* so that the terms in the different ontologies are made to correspond. Sometimes these mappings are only partial, because the ontologies, developed for different original purposes, might not have needed to define all the same concepts or might have defined them differently. The descriptions of such mappings are sometimes called *articulations*^{5,6} or *bridging axioms*⁷ and can include simple correspondences between terms (uni- or bidirectional), rules defining terms in one ontology relative to some set of terms in another, and even functional mappings, such as for translating units of measure.

The book-buying example in Figure 1 schematically illustrates the issues. I, as a user, have a personal shopping agent (MyShopper) that can assist me in finding and buying many different kinds of items. It does this by using Semantic Web Services registries, or "matchmakers," to help it identify reputable sources for each new kind of item I request and then interacting with those services to

complete the purchases. In addition to knowing about services, my agent knows many things about me and my possessions, including my credit cards and other personal information useful to its tasks. MyShopper finds a service called CongoBooks, which has published a service model using OWL-S. The service model declares that if the **BuyBook** process is invoked successfully, which requires the book be in stock, its *effect* will be that the requester owns the specified number of copies of the requested book. The process description specifies as *inputs* the book title, author, quantity, and hardcover or softcover preference. All these things are described in the ontology that service uses. It produces as output, if successful, a message containing an order confirmation number and a shipping tracking number (not shown in the figure).

At first blush, it seems that MyShopper 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 CongoBooks' process descrip-

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

MyStuff	Bridging relationship	Inventory
Book	<i>InstClass-SetClass</i>	Item
Book.name	<i>equivInstanceProp</i>	Item.title
Book.by	<i>equivInstanceProp</i>	Item.author
Book.pubDate	<i>equivInstanceProp</i>	Item.PubDate
Book.purchDate	—	
Book.shelfLocation	—	
		Item.qty

tion. However, the goal of owning a book corresponds not to the description of the process and its inputs, but to a stated effect of the service process description, specifically that the buyer owns whatever item was requested when the process completes successfully.

A classic AI planning system would rely on just such a correspondence between goals and operator effects, and MyShopper can plan to achieve its goal by instantiating the single action represented by this service. That is, by treating this service description essentially as a *planning operator*, MyShopper can build a plan that uses the CongoBooks **BuyBook** process as an abstract model of a message-passing action for which it must determine the correct inputs. It executes the resulting plan by using the OWL-S service *grounding* to translate these inputs (as specified in the service’s ontology) into a message in WSDL, and transmitting it.

Overall, OWL-S was designed to support 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 service processes, and a way to associate these processes with WSDL message templates. MyShopper can thus reason that it will succeed if it can invoke the **BuyBook** process with a message containing the required attributes (title, author) of the desired book, along with other required process inputs. I can expect the process to achieve my goal because the process description says the item owned when the process completes will have the same attributes as the book in my goal if I specify those attributes as inputs.

Request formulation with integrated translation reasoning

Several points in the agent’s chain of reasoning might require translation. OWL-S’s use model is based on the model suggested

for reading documents on the Semantic Web. That is, clients who want to use unfamiliar services can reason how to interact with them by first reading their published service descriptions, which will naturally involve finding and loading the ontologies those descriptions are based on. In this case, the process model and ontologies can be read but only partly understood, because some of the ontologies will be unrelated to those the client uses internally. As Figure 1 illustrates, the two agents shared the ontologies for OWL-S itself and for basic economic transactions (econ101), including the **Owns** relationship. However, the agents have different representations of books and credit cards.

MyShopper can unify the **Owns** effect of the **BuyBook** process with its goal, but to then identify the **title** and **author** inputs to the process, it must be able to find the correspondences between its concept of **Book** and the service’s concept of **Item**. For example, CongoBooks requires that the input identifying the **title** of the purchase item be the same as the **title** of the item mentioned in the **Owns** effect of the process model. To fill this input requirement, my agent needs to reason deductively using its own knowledge base to identify a value for the **title** property for the **Item**, referenced in the effect. MyShopper uses the MyStuff ontology—which includes the concept **Book** with properties **name**, **by**, **pubDate**, **purchDate**, and so on—for representing books in its collection. To do this translation reasoning, MyShopper needs a set of bridging axioms⁷ that define the relationships between terms in this ontology and terms in the ontology CongoBooks uses. Assuming that this articulation exists as shown in Table 1 in a published mapping relating the two ontologies, MyShopper can conclude that things of type **MyStuff:Book** can be equated with things of type **Inventory:Item**. It can also conclude that the **name** property of **Book** can be related to **Item.title** as part of the normal goal-regression reasoning you asso-

ciate with an automated planner.

This works despite the fact that the relationship between the two ontologies isn’t simply one of term equivalence (see Table 1). In the MyStuff ontology, elements of the class **Book** are specific physical objects sitting on particular shelves, where in the inventory model CongoBooks uses, each **Item** description represents a quantity of like items the company has in stock. Nonetheless, these two concepts can be related, as some of the descriptive properties of the class **MyStuff:Book** are equivalent to descriptive properties of **Inventory:Item**. For example, each instance of type **Item** represents a number of books, all of which have the same **title**, **author**, **firstPubDate**, and so on. For this reason, we can use bridging axioms that capture the conditions under which an **Inventory:Item** description (with quantity = 1) can refer to the same entity as a description of a **MyStuff:Book**.

Identifying non-goal-related inputs

So far, we’ve seen how an agent can identify several of the required inputs to the **BuyBook** process by reasoning about the relationship between the client’s goal and the effects of a published process description from an unfamiliar service provider. The identification of these inputs hinges on two things:

1. The client can match its goals to (a translation of) some of the proposed process’s effects, unifying items referenced as arguments to these goals with the variables in those effects
2. All type restrictions, effects, and preconditions referencing these input variables 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 first step is to unify the process’s effects with the client’s goal and then satisfy the process preconditions. The planner’s unification process can succeed only if the effects can be translated into the client’s native ontology. The input and local parameters act like variables in a planning operator, while the translated effects and other constraints that reference inputs are the class and property relationships that must be used to identify values for those variables. Incrementally translating these propositional elements from the service provider’s ontology to the client’s ontology forms a natural extension of the client planner’s reasoning.

Unfortunately, even if the mappings are available to do this successfully, the reasoning I just described isn't generally enough to determine all of the inputs to a service. There's also the question of providing required inputs that aren't part of the client's explicit goals. Processes often require inputs whose effects don't relate directly to elements of the client's goals but can nonetheless be determined directly on the basis of the client's knowledge or as a result of additional decisions the client made.

For example, consider the credit card and shipping information inputs in our sample book-buying process. One effect of the **Buy-Book** process is that, when successful, a book ships to the address specified by input **shipto**. Let's assume that MyShopper was designed to handle this using a default rule that all purchases requiring shipment be shipped to the buyer's address. Of course, exceptions such as gift purchases could occur, in which case, the requirement would be specified as an additional goal, which could be handled by planning shipment separately, if necessary (such as when the service doesn't ship internationally). The point is that services will vary in how they address these secondary effects. The client must handle them as they arise, when confronted with a new service description it intends to use. To do this, the client must, be able to make choices about the preferred dispensation of effects that are not its primary intention.

The translation issue here is that such background policies or preferences can be applied only if MyShopper can identify the effect's mapping (**Shipped ?item ?addr**) in the CongoBooks ontology to a corresponding description in a preference or default rule MyShopper has internally. This again argues that the effects and other constraints on the process's input variables must be translated back into the client's ontology, so that the client can determine permissible input values.

Another effect of the process is that a credit card is debited by the purchase amount (plus taxes). Indeed, the process should also have a precondition that the credit card identified by the inputs has sufficient credit available for that effect to occur. Suppose that MyShopper has in its knowledge base that the user has three credit cards with varying levels of available credit, one of which is only for business purposes. To invoke the **BuyBook** process, MyShopper must decide which credit card to use for the purchase. MyShopper can make this decision in consultation with the user or

entirely on the basis of a set of policies in combination with the precondition constraints (such as "not expired" or "enough available credit") specified by the service provider in the published process description. Even if the user were involved in the decision, the choices could first be narrowed on the basis of the service's restrictions, such as when the service provider didn't honor certain cards.

These examples reinforce the point that one critical translation issue for service invocation is translating service input constraints into the client's ontology so that the client can perform necessary decision-making. The process isn't simply one of finding a credit card number and an expiration date and plugging them in. These values must be associ-

A critical translation issue for service invocation is translating service input constraints into the client's ontology so that the client can perform necessary decision-making.

ated with a single credit card of the right type and with an appropriate available balance.

The advantage of translation as extended reasoning

We've suggested that perhaps the best way to effect the needed translation is to make available the ontology mappings or bridging axioms and let the client use them during its reasoning, much the way a nonnative speaker might try to fill in a Web form written in another language using a translation dictionary. The alternative is to use a third-party translation service to translate the service description before the client reads it. Although this is indeed a viable alternative that could use the same set of mapping rules, several caveats exist. First, the translation isn't of a message sent from another agent, but of a published document about such an agent. Second, although the client can do the translation on an as-needed basis during its goal-directed reasoning, the translation agent will attempt to translate the whole process model.

This might be both unnecessary and lead to problems when the mapping rules relating the client's and server's ontologies are inadequate to perform a full translation. For example, when attributes of no consequence to the client are represented in the server's ontology, the translation agent might fail. In our Figure 1 example, CongoBooks has an input that distinguishes the hardcover and softcover editions of books, a distinction the designers of the MyStuff ontology didn't make. Other elaborations of this use case might include input parameters that are relevant only when a client wishes to buy items other than books but for the service provider are still represented using the same **Item** class. If such problems cause the translation service to fail entirely when it can't complete a translation, then the client won't be able to use the service, for irrelevant reasons.

Another issue is that the client must, in the end, represent the required inputs in the service provider's ontology so that they can be correctly grounded in a WSDL message. The grounding the service provider gives is specified in terms of those ontologies. This means that after inputs are identified they must be translated back into the target service's representational formalism, and then serialized for inclusion in a WSDL message template. Again, this could be handled by dispatching to a third party translator, but unless that middle agent was also going to apply the grounding, the result of translating each individual input would have to be passed back to the client for the WSDL message's final packaging. At best this model is very awkward.

A final advantage of published mappings and clients that can do their own translations is that these agents, which must interact with their users in any case, can enlist these users to assist with decisions about unanticipated auxiliary input preferences and, potentially, the translation (or even the mere relevance) of input parameter constraints. For the foreseeable future, ontologies that different communities use will be subject to the problem that mappings between them will at best be partially complete. In practice, we'll need to enlist both users and ontology builders in the incremental accumulation of ontology mappings that gradually increase the interoperability of our systems.

Integrated planning and translation

Drew McDermott and I are implementing the approach I've described that interleaves

planning and translation for invoking Semantic Web Services. We used OpTop,⁸ an estimated-regression planner developed at Yale, for this, because the reasoner used for goal regression can easily accommodate the first-order rules representing the required bridging axioms. This joint work extends and combines our approaches to Web Service composition planning⁸ and ontology translation.⁹ The translation method is based on first merging ontologies while keeping their terms distinct and then introducing bridging axioms that relate the terms. The Semantic Web allows ontologies to be combined naturally and interrelated in just this way by using URIs as namespaces to distinguish the terms of different ontologies.

As I suggested previously, we make the additional assumption that for interoperability between intertranslatable ontologies, bridging axioms defining correspondences between terms in different ontologies must also be openly shared on the Web. In this article, I've assumed that these bridging axioms exist and are available to the client reading the service models of unfamiliar services. So, the question becomes one of ensuring the mappings exist. Clearly, sometimes these bridging rules will be incomplete or absent, and a human will need to intervene to define additional correspondences before the agent can reason with them. We believe that as a practical matter, exception handling associated with partial translation will be extremely important. This will enable human users of client systems to be able to contribute to extending the mappings to cover new cases at runtime. Such incremental learning of these relationships will lead to more robust behavior in the long run, as long as there is a process for validating the new information.

As the CongoBooks example suggests, 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 by querying an OWL-S Matchmaker,¹⁰ the client must perform all of the following:

1. Read the service process model, loading all ontologies referenced in 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 preconditions and type constraints on input variables in the service model into terms based in its own ontologies.
4. Unify the effects of the process with the current planning goals and preferences, as expressed in its local ontologies. This is a normal step in the planning.
5. Select bindings for unbound input variables that satisfy all the operator preconditions and type constraints and are consistent with the effect bindings.
6. Translate the descriptions unified with input variables into the service provider's ontology.
7. Apply the process grounding to map these inputs into the format of the

For interoperability between intertranslatable ontologies, bridging axioms defining correspondences between terms in different ontologies must be openly shared on the Web.

specified WSDL request message, then send it.

Steps 1, 4, 5, and 7 have been implemented in systems that use AI planners to compose and execute DAML-S or OWL-S service process models.^{4,11-13} Steps 2, 3, and 6 are new steps required to do this when the client's ontology differs from the service provider's.

Steps 2 and 3 together describe the problem of using preexisting 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 might have been developed independently. The critical elements to be translated are the domain-specific elements implicated in reasoning by the planner: the types of objects (*Item* -> *Book*, for example) and the relationships used to identify or constrain the use of those types of objects in processes.

We treat the translation process as an application of bridging axioms in either a forward- or backward-chaining direction.^{9,14} The result is a set of projections of terms from one ontology into terms of a target ontology. The key test for such projections for this application is whether all the constraints represented in a service process model have been successfully translated so that the critical planning decisions can complete. We currently use mappings developed using an interactive tool and published using an RDF encoding of first order logic rules that's compatible with the extended Planning Domain Definition Language (PDDL) the OpTop planner uses. We're evaluating whether the newly proposed Semantic Web Rule Language (www.w3.org/Submission/SWRL/) could also be used to describe these mappings.

Step 6 amounts to translating the request into a form that enables it to be communicated. In essence, this is what has traditionally been conceived of as the role of a mediator. Once the planner has proved to itself that the instantiated 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 trivial. But suppose, for example, the BuyBook process required a "deliver by" date, and the client represented dates as complex objects with a month, day, and year, while the service ontology represented them as a fixed format strings. This transformation would have to be performed to instantiate the operator.

As a result of these issues, we believe request message formulation based on published process models requires tightly coupled support for ontology translation during the client's planning. The client can't simply formulate a message on the basis of its goal and then have the message translated, because the client must interpret the service model to do the plan reasoning required to establish the request message's necessary elements. So, request formulation for published service process models is most easily provided using a planner that can reason directly with bridging axioms while establishing operator preconditions and variable value assignments for the operators represented by those process models. In such cases, the client transparently performs the translation, using the merged ontologies' bridging axioms as it reasons

about how to satisfy the preconditions of the process to be invoked.

Translating responses to service requests

Of course, sending a request to a server is only half the battle. The response must also be interpreted. This, too, requires translation, though of a more traditional kind. Here, the information to be provided is available as a coherent message the sender or service provider developed, and the issue is identifying a mapping of this message into the client recipient's ontology. The OWL-S grounding model, as provided with the service process model, specifies how to map the produced WSDL message into a set of semantically described outputs represented using terms in the service provider's ontologies. To translate these outputs into descriptions useful to the client, the agent that does the translation must first apply the grounding, then the bridging axioms. If both the grounding and the bridging axioms are published, then response translation can be done by any agent that has access to the source and target ontologies and to the necessary mapping rules. However, because the service provider might not actually work directly with the information in its logical form, there is no reason for the service provider to do the work. The client has already loaded the service model and the mapping rules, so it is able to do this quite effectively. The alternative is for a middle agent to be enlisted to apply the grounding and translate the result.

Many researchers working on heterogeneous information retrieval have addressed the problem of developing the necessary ontology mappings^{15,16} and support for the translation of well-formed communications or data query responses.^{5,9,17,18} As a practical matter, the strategies used for assigning the task of doing the translation depend on the size and variability of the messages to be translated. For example, we've previously described an approach to compiling out message translations that works well for large data sets that are exchanged regularly by generating code for translating specific classes of structured data into an alternative new form.¹⁷ The approach assumes that both agents' ontologies are available, along with axioms that relate the two and a target data pattern. When an agent A is to answer a query Q by agent B, but the two use different ontologies, our translation middle agent would develop a specific translator for the class of

data sets D_A that include the query response, as represented using ontology O_A . The generated code would translate these data sets into data sets of class D_B , represented in the target ontology O_B . This does not work as well with Web Services or agents that can generate very different responses to a request, depending on their internal state. For example, a purchase request might result in a confirmation message or an "out of stock" message. Or, a query to an agent containing an RDF store might result in a response consisting of a heterogeneous list of object descriptions. The approach McDermott and I have taken for such Semantic Web agents employs a first-order rule engine to translate RDF and OWL descriptions.⁹ Similar tech-

Mapping servers, in addition to translation servers, will be needed, because the Semantic Web doesn't have an effective way of indexing or accessing information that relates several different ontologies.

niques exist,^{5,19} although the method Hans Chalupsky describes uses more syntactic transformation rules.

Examples such as those I've presented show why, as a practical matter, service requests can't be formed and then translated, because most of the translation is done when interpreting the provider's service description. If a service requester must plan how to formulate its requests, then it has the context necessary to translate the published service description as needed for that reasoning. More generally, as I've argued elsewhere, the question of *who* (which agent) should translate particular semantic representations associated with dynamic Web Service interactions will depend on the specific processing goals and amount of background knowledge that's local to each particular class of agent (sender, recipient, middle agent).²⁰ For example, matchmakers that can accept and store

service advertisements developed using different ontologies will likely be ineffective unless they can use ontology mappings while matching queries to potential candidate services. Service discovery queries, as individual descriptions, must use a single ontology or consistent set of component ontologies that will be different from at least some of those used for advertisements known to the matchmaker. The general point is that multiple different agents will need to do translation for different parts of the overall process, so it's imperative that the mappings be published such that they can be discovered when the need to translate arises. This means that mapping servers, in addition to translation servers, will be needed, because the Semantic Web doesn't presently have an effective way of indexing or accessing information that relates several different ontologies.

This article has focused specifically on how service clients will need to use ontology mappings to invoke unfamiliar services. I haven't addressed here how the clients find such services or select among them. In fact, for many applications, such as composing grid services, users might choose the services themselves and leave it to the client agent to determine how to interact with them. For others, matchmakers and peer recommendations might be used. In many of the latter cases, clients might have to negotiate directly with candidate services to determine which best meet their needs and agree to a contract meeting the detailed requirements of their particular use (such as price, time of service, and so on). Although beyond this article's scope, these negotiations will also require some amount of translation support when dealing with unfamiliar services. ■

Acknowledgments

I thank my colleague Drew McDermott of Yale University and the other members of the DAML-S Consortium for discussions on this and related topics. A contract with Yale University to the DARPA DAML program (contract F30602-00-0600) funded this writing of this article.

References

1. M. Dean et al., "OWL Web Ontology Language 1.0 Reference," 2003; www.w3.org/TR/owl-ref.
2. DAML Services Coalition (A. Ankolekar et al.), "DAML-S: Semantic Markup for Web Services," *Proc. Int'l Semantic Web Working Symp. (SWWS)*, 2001; www.semanticweb.org/SWWS/program/full/paper57.pdf.

The Author



Mark H. Burstein is a division scientist and the director of the Human Centered Systems Group at BBN Technologies. His research focuses on AI and cognitive science, including control of

multiagent systems, memory-based reasoning for natural language processing, cognitive models of human plausible and analogical reasoning and learning, knowledge acquisition, mixed-initiative planning and scheduling, semantic translation, and Semantic Web Services. He's cochair of the Semantic Web Services Initiative's Architecture Committee and a founding member of the DAML Services Coalition. He received his PhD in computer science (AI) from Yale University. Contact him at BBN Technologies, 10 Moulton St., Cambridge, MA 02138; burstein@bbn.com; <http://daml.bbn.com/burstein>.

3. E. Christensen et al., "Web Services Description Language (WSDL)," 2001; www.w3.org/TR/wsd1.
4. M. Paolucci et al., "The DAML-S Virtual Machine," *The Semantic Web-ISWC 2003: 2nd Int'l Semantic Web Conf.*, LNCS 2870, Springer-Verlag, 2003, pp. 290-305.
5. P. Mitra, G. Wiederhold, and M. Kersten, "A Graph-Oriented Model for Articulation of Ontology Interdependencies," *Advances in Database Technology: EDBT 2000*, LNCS 1777, Springer-Verlag, 2000, pp. 86-100.
6. A. Maedche and S. Staab, "Semi-automatic Engineering of Ontologies from Texts," *Proc. 12th Int'l Conf. Software Eng. and Knowledge Eng. (SEKE 2000)*, 2000, pp. 231-239.
7. D. McDermott, M. Burstein, and D. Smith, "Overcoming Ontology Mismatches in Transactions with Self-describing Agent," *The Emerging Semantic Web: Selected Papers from the 1st Semantic Web Working Symp.*, IOS Press, 2002, pp. 228-244.
8. D. McDermott, "Estimated-Regression Planning for Interactions with Web Services," *Proc. 6th Int'l Conf. Artificial Intelligence Planning Systems (AIPS 2002)*, AAAI Press, 2002, pp. 204-211.
9. D. Dou, D. McDermott, and P. Qi, "Ontology Translation on the Semantic Web," *On the Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, LNCS 2888, Springer-Verlag, 2003, pp. 952-969.
10. M. Paolucci et al., "Semantic Matching of Web Services Capabilities," *The Semantic Web-ISWC 2002: 1st Int'l Semantic Web Conf.*, LNCS 2342, Springer-Verlag, 2002, pp. 333-347.
11. D. Wu et al., "Automating DAML-S Web Services Composition Using SHOP2," *The Semantic Web-ISWC 2003: 2nd Int'l Semantic Web Conf.*, LNCS 2870, Springer-Verlag, 2003, pp. 195-210.
12. M. Sheshagiri, M. desJardins, and T. Finin, "A Planner for Composing Services Described in DAML-S," *Proc. AAMAS Workshop Web Services and Agent-Based Eng.*, 2003.
13. E. Sirin, B. Parsia, and J. Hendler, "title here," *IEEE Intelligent Systems*, vol. 19, no. 4, 2004, pp. XX-XX.//RITA: UPDATE REFERENCE WIT ARTICLE INFO//
14. D. Dou, D. McDermott, and P. Qi, "Ontology Translation by Ontology Merging and Automated Reasoning," *Proc. 13th Int'l Conf. Knowledge Eng. and Knowledge Management: Workshop Ontologies for Multi-Agent Systems*, 2002.
15. E. Rahm and P.A. Bernstein, "A Survey on Approaches to Automatic Schema Matching," *VLDB J.*, vol. 10, no. 4, 2001, pp. 334-350.
16. Y. Kalfoglou, and M. Schorlemmer, "Ontology Mapping: The State of the Art," *Knowledge Eng. Rev.*, vol. 18, no. 1, 2003, pp. 1-31.
17. M. Burstein et al., "Derivation of Glue Code for Agent Interoperation," *J. Autonomous Agents and Multi-Agent Systems*, vol. 6, 2003, pp. 265-286.
18. J. Madhavan et al., "Representing and Reasoning about Mappings between Domain Models," *Proc. 18th Nat'l Conf. Artificial Intelligence (AAAI 02)*, AAAI Press, 2002, pp. 80-86.
19. H. Chalupsky, "OntoMorph: A Translation System for Symbolic Knowledge," *Proc. 7th Int'l Conf. Principles of Knowledge Representation and Reasoning (KR 2000)*, Morgan Kaufmann, 2000.
20. M. Burstein, "The Many Faces of Mapping and Translation for Semantic Web Services," *Proc. 4th Int'l Conf. Web Information Systems Eng. (WISE 03)*, IEEE CS Press, 2003, pp. 261-268.

Look to the Future

IEEE Internet Computing reports emerging tools, technologies, and applications implemented through the Internet to support a worldwide computing environment.

In 2004-2005, we'll look at

- Wireless Grids
- Measuring Performance
- Homeland Security
- Internet Access to Scientific Data
- Recovery-Oriented Approaches to Dependability

... and more!

IEEE
Internet Computing

www.computer.org/internet

