Draft of article in AI Magazine 26(1) : 71-82, Spring 2005. Final version is copyright AAAI.

Ontology Translation for Interoperability Among Semantic Web Services

Mark H. Burstein and Drew V. McDermott

BBN Technologies
Cambridge, MA
burstein@bbn.com

Yale University
New Haven, CT
mcdermott@cs.yale.edu

Abstract

Research on semantic web services promises greater interoperability among software agents and web services by enabling content-based automated service discovery and interaction, utilizing shared ontologies published on the semantic web. However, services produced and described by different developers may well use different, perhaps partly overlapping sets of ontologies, just as it is common that relational database schemas for similar or related functions are only partly compatible. Once again, interoperability will depend on ontology mappings and architectures supporting the associated translation processes. The question we ask is, will the traditional approach of introducing mediator agents to translate messages between requestors and services work in such an open environment? This article reviews some of the processing assumptions that were made in the development of the semantic web service modeling ontology OWL-S [OWL-S] and argues that, as a practical matter, the translation function cannot always be isolated in mediators. Ontology mappings will need to be published on the semantic web just as ontologies themselves are. The translation for service discovery, service process model interpretation, task negotiation, service invocation and response interpretation may then be distributed to various places in the architecture, so that translation can be done in the specific goal-oriented informational contexts of the agents performing these processes. We present arguments for assigning translation responsibility to particular agents in the cases of service invocation, response translation and matchmaking.

Introduction

Semantic Web Services are to the Semantic Web what current-day web services are to the web as we know it now. Tim Berners-Lee, Jim Hendler and Ora Lassila's vision of the Semantic Web [TBL01] was of a future web populated by pages enriched by their association with sharable semantic representations, which describe both content and, in the case of services, functionality. Semantic web services will publish machine interpretable descriptions of their capabilities and interaction models so other software agents can find and use them without prior 'built-in' knowledge about how to call their APIs. They will soon support the development of personal software agents or 'semantic web clients' for such things as comparative shopping, information discovery and travel planning, and compositions of those services. Less glamorously, but perhaps more importantly, these techniques may soon enable business-to-business interactions that are more dynamic, support semi-automated service composition on the scientific computing Grid and enable mobile, wireless devices to be able to interact seamlessly with the services discovered as they move about.

To fulfill these promises, published semantic service descriptions must be used in a variety of ways. Services will be *discovered* by agents matching client's service requirements against service capabilities. Clients will *invoke* services by deducing from their descriptions the content of the messages required to request those services and interpret their responses, which may range from straightforward acknowledgements to indications of failure to requests for additional information. Finally, by using these descriptions of each service's effects and usage constraints, agents may compose multiple services, roughly the way classical AI planners use planning operators. We use the word *agent* for these clients to emphasize the goal of giving them the ability to reason about the services they deal with.

Internally, semantic web service clients must be able to determine when to "outsource" an internal goal or function to a remote service, select among some suggested candidate services and reason about *how* to interact with the selected service based on the service's published description and their own internal goals and knowledge. This includes decisions about how to provide ancillary information the services may require, (e.g., credit information, access certificates, etc.) Since services may require extended interactions, service descriptions may include interaction protocols that these clients must be able to follow. A noteworthy example is failure-recovery procedures.

A number of auxiliary semantic web agents can help in achieving these aims. Semantic *matchmakers* [Pao02a] that act like web search engines or intelligent UDDI [UDDI] web service registries may assist with automated service discovery by cataloging and recommending services to clients. Authentication and policy authorization services may assist both clients and servers to know who they are, and what kinds of interactions they can have. Ontology and mapping registries may help to ensure that agents have consistent and complete sets of concepts, relations and rules for service-related reasoning.

Semantic service descriptions are developed using a mix of domain ontologies and shared, general-purpose ontologies, such as those defining the structures for representing service capabilities. OWL-S [OWL-S], formerly DAML-S [Da02], is a semantic web ontology developed by a group of researchers in DARPA's DAML program, to address these latter, structural aspects of service descriptions. The World Wide Web Consortium (W3C) has formally recommended the Resource Description Framework (RDF) [RDF] for building web-compatible structured semantic descriptions and the Ontology Web Language (OWL) [OWL] for publishing ontologies (web documents) defining structured classes and relations. OWL is a semantic description language with a formal semantics and an XML/RDF syntax. By leveraging features of RDF, OWL ontologies can be shared, combined, and extended simply by publishing web documents. OWL-S is a set of ontologyies, developed in OWL, for writing machine readable semantic descriptions of web services such as those whose APIs are described using the Web Service Description Language (WSDL) [WSDL].

In this article, we review the basic elements of OWL-S and its intended use model, and then discuss the role that ontology mapping and translation must play in interactions between clients and services that use different domain ontologies. We will argue why, in practice, we expect particular agents will need to be responsible for translating the content of messages produced at different stages of their interaction, and why it may at times be difficult for mediators to relieve the functional agents (services and clients) of this responsibility.

Communities Sharing Ontologies *and* Ontology Mappings on the Semantic Web

Although this article is about semantic translation of service requests, OWL-S is useful even if clients and services use the same ontologies. Clearly, service interactions are simpler when one doesn't have to translate the meaning of messages sent between agents, so one might ask why not just share a single consistent set of ontologies? When ontologies are shared within a community of people and software services, then major barriers to efficient and timely information sharing are removed. This is why businesses and governments have spent millions of dollars in recent years trying to unify ontologies and database schemas to the largest extent possible. Unfortunately, these efforts have also demonstrated that there are also costs associated with ontology or schema merging that grow with the number of disparate systems and modeling perspectives being combined. At some point, the level of detail in the unified representational model is greater than that needed by any of the individual applications, increasing the complexity of maintaining all of them. The alternative, as suggested by Uschold [Usch02], among others, is to share ontologies within tightly integrated communities, while allowing for mediated interaction with other communities. Each community can develop some of their own ontologies, and share or extend other widely shared concepts, maintaining locally more detailed models within communities with special responsibilities.

When two communities are going to interact, their ontology developers must define (at least partial) mappings between the ontologies each uses, and extend the original ontologies where needed so that messages between them can be translated into the recipients native ontology. Although there is a sizable literature on automated and semi-automated techniques for developing mappings (See [Rah01, Noy02, Kal03] for surveys), we anticipate it will require at least some involvement of people familiar with each of the ontologies involved for some time to come. RDF and OWL are designed to enable semantic web communities to share a syntactic language for defining ontological terms and communicating semantic descriptions. We submit, though, that *ontology mappings*[1] will also need to be published using a similarly standardized language, and agents will also need to interpret these mappings when communicating with 'foreign' agents. OWL, as a terminological description language, is not powerful enough to describe all of the kinds of mappings between the concepts that it can be used to define. In our own recent work, we have used first-order logical rules [Dou02, Dou03] to define these mappings, and previously explored the controlled application of second order rules to generate translation *programs* [Bu03b]. In this paper, we focus on the question of *how to use* mappings, once published, to interact with services that use different ontologies.

The OWL-S Ontologies for Semantic Web Services

OWL-S is a collection of "upper ontologies" for describing web services from two primary perspectives, advertising/discovery and planning/execution. It does not address (or need to address) specific domain issues (e.g. product

[1] See [Mad02] for a discussion of ontology mapping languages.

taxonomies, taxonomies of types of inputs, etc.) as these ontologies can developed, shared and adapted by communities of service providers. OWL-S' service models consist of three abstract components. The *service profile* ontology is used to describe service 'advertisements' suitable for use by a semantic web service matchmaker. It includes an extensible set of properties for describing the purpose of a service (as defined in a service hierarchy), its inputs types, output types, and other features that might be relevant discriminators between services from the perspective of potential users of that service.

An OWL-S matchmaker [Pa02a] accepts advertisements from services and queries from clients looking for services. It finds candidate services for clients by matching queries described by partial service profiles against the profiles registered by services[1]. Several OWL-S matchmakers have been developed, including one that extends the web service registry system UDDI [Pao02b].

The OWL-S *service process model* includes concepts for describing in detail the interfaces and functional properties of *atomic* services, namely, their inputs, preconditions, conditional outputs and effects. It also includes a vocabulary for process composition so service providers can describe how clients should execute sequences of services and their associated data flow. The most recent release of OWL-S treat process specifications as named individuals with complex descriptions, in a manner consistent with the activity model in the Process Specification Language (PSL) [Sch00]. Pre-conditions, post-conditions and effects are represented using encapsulated logical expressions.

The OWL-S process model ontology is also based in part on traditional representations of classical AI planning operators, as exemplified by the PDDL language used in the annual AI planning competition [Fox03]. The premise here is that the reasoning required to identify and invoke an appropriate service is essentially the same as selecting from among a set of planning operators to achieve a goal, and then executing it. *Input parameters* represent the information required to invoke the service, and *output parameters* represent the information that is returned by the service. *Preconditions*, and *conditional* service *effects,* whose ranges are logical formulas, are to be interpreted by the client reading the service description as additional constraints on the inputs it must provide, with the effects also describing how the service changes the state of the world in a way hopefully consistent with the client agent's goals. Many planning systems have been used to implement OWL-S service clients, including [McIl02, Pao03, Wu03].

There is good news and bad news about applying traditional planning algorithms to the semantic web service problem. The bad news is that the assumption that the state of the world is completely known at all times is obviously wrong. In fact, a large fraction of the actions taken on the web are to gain information [McD02a]. The good news is that there are a number of planners that can handle simple contingent planning with partial state knowledge and information acquisition goals (See survey by Weld [We99]), and the standard notation for action definitions actually fits the web pretty well. Actions are invoked and monitored by messages, and the effect of a message often does consist of changing a few things discretely. Our own work has been based on extending the estimated regression planner, Optop [McD02b, McD03], to do contingent planning with OWL-S process models.

The last part of the OWL-S model, the *service grounding,* describes the relationships between the inputs and outputs of atomic processes and the elements of a particular message transport model. Specifically, the grounding ontology represents mappings of process parameters onto portions of WSDL message specifications. The OWL-S notion of an *atomic process* can be viewed as an abstract representation of a WSDL operation that provides additional semantics for it. Although WSDL is rapidly becoming the standard XML language for describing web services, it has no internal means for specifying the semantics of a services message patterns. WSDL message parameters have XML datatypes, whose semantics must be interpreted by the client program, by virtue of its programmer having read the accompanying documentation.

The OWL-S ontology thus adds the semantic elements necessary for agents to reason *dynamically* about the relationship between their internal goals and the types of information required to formulate service requests and interpret service responses. Once the *information* required to create a request has been identified, OWL-S provides a mapping to WSDL (the grounding) to enable the request to be turned into a SOAP or HTML format.

OWL-S Usage Model

The basic use model envisioned for OWL-S, illustrated in Figure 1, consists of the following steps, not all of which are required in all cases. Services first advertise themselves by sending their service profiles to a matchmaker and publishing service process models on the web so clients can read them (1). Agents requiring a service to achieve *goals* pose queries to

[1] This matching process is not a straightforward subsumption test, as one might expect. As [Pao02a] shows, it is asymmetrical. A good match is one where the inputs the client can subsume those required, while the outputs the service claims it can return subsumes those required by the client.

the matchmaker consisting of a (partial or abstract) description the service it requires, including non-functional or quality of service requirements (2). The matchmaker compares these abstract descriptions to its library of service profiles and returns (URI references) for candidates that could be used (3). Each agent then reads the candidates' published service descriptions and selects one to use. The agent determines how to make a request to the selected service by unifying its goals and preferences with the effects specified in the service process model, in order to determine a semantically valid set of inputs to the server. These inputs are then mapped onto a WSDL input message pattern using the OWL-S service grounding, and the resulting message is sent (6).

The service receives the request, and determines whether it can perform the request. It may acknowledge the request, send a error, request additional information, or (generally, on completion,) send a reply stating the service results (7). Whatever reply is sent is parsed using a WSDL error or output message template. This message is then mapped (using the process output grounding) into a semantic descriptions of the output parameters of the OWL-S process, from which the agent determines which of the published process' effects occurred and whether the agent's goal has been accomplished (8).

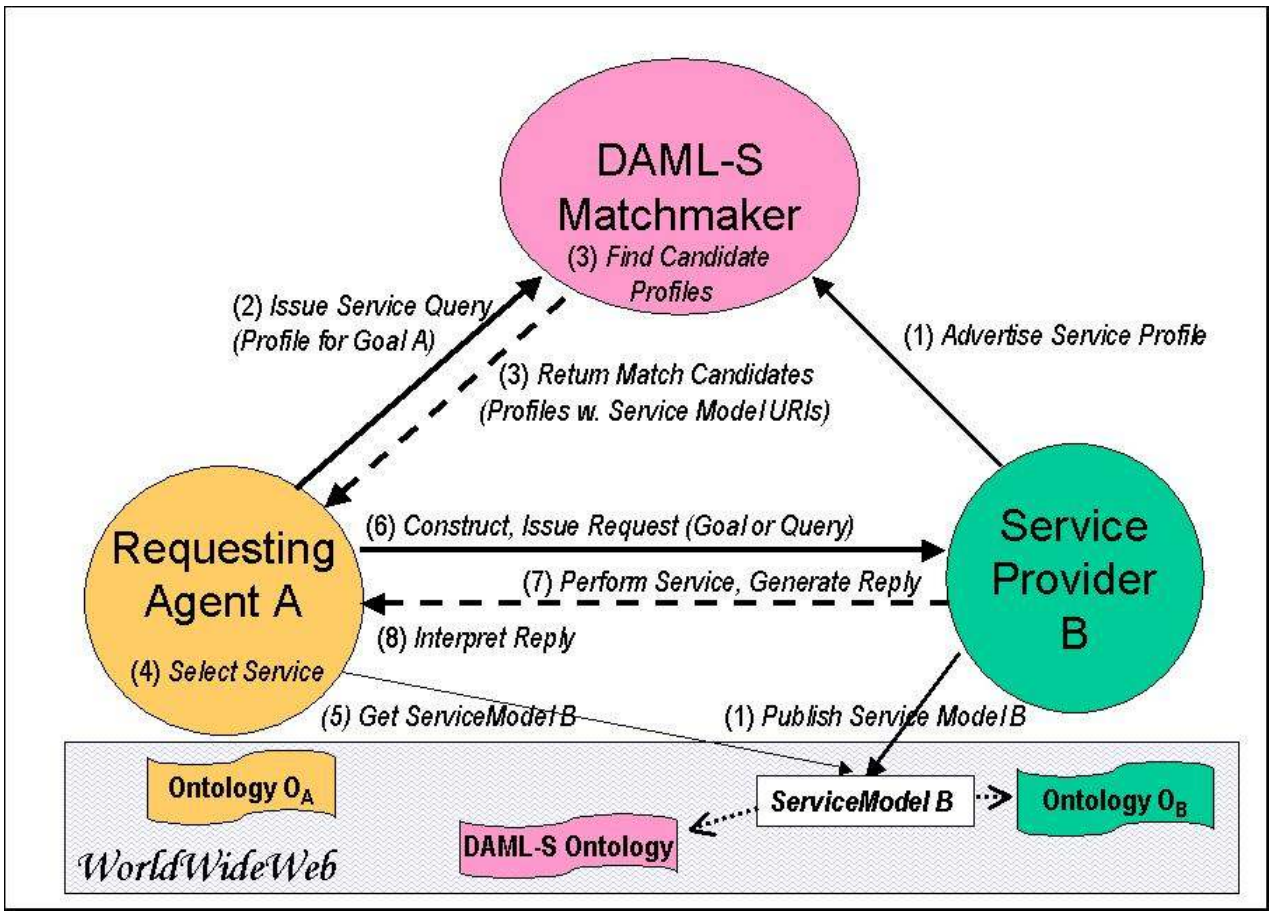

Figure 1: Basic OWL-S Semantic Web Service use model.

Viewed abstractly, this process bears some resemblance to the web service use model envisioned for UDDI-WSDL-SOAP, except that it is does not involve a programmer querying UDDI, reading the WSDL models found there, and implementing those interfaces. The client software agent is responsible for the interaction with the matchmaker, the interpretation of which candidate services are most appropriate, the determination of the information required to invoke each service, and the interpretation and response to messages returned by the service.

Variations on this execution model for OWL-S have been implemented by a number of academic and industrial researchers (e.g., [Gai03, Pao03, Sab03, Sir04]). These implementations have finessed the ontology translation issue by assuming a shared set of ontologies. However, as suggested above, we cannot assume that agents always share ontologies. Translation

will be required to achieve the broad interoperability envisioned for semantic web services, and so we need an architecture for that supports it.

Service Invocation Reasoning Across Different Ontologies

The central questions we will discuss here are: where does translation fit into an architecture supporting the OWL-S use model, and how do the goals and knowledge of specific agents influence the decision about which agents can or should do translation? The traditional approach to inserting translators into distributed architectures is to insert a middle agent between client C and server S to translate messages from language L_C into language L_S and vice versa. We will argue that this only works when the sender can construct a well-formed message, and the target ontologies and ontology mappings are known. It doesn't work, or work well, when a translator needs information that is local to a client or service provider, rather than just the ontologies used by the agent receiving the translation.

Consider the difference between a client's internal goal and a well-formed request to a service. For example, to request a purchase, a service may require specific information about the requestor (e.g., login id, valid means of payment), and the manner of accomplishing the goal (e.g., shipping method). The request message must be constructed dynamically by reasoning about the relationship between internal client goals and the required input parameters specified in a published service description. The first translation task is therefore the translation of domain elements of the service description from the ontologies used in the published service description into ontologies used by the client. We have developed a model showing how the client can handle this translation task as part of its service request planning reasoning, by reading the service description in its original form along with the ontology mappings needed to interpret the constraints on each information requirement. In an environment with all of these axioms, its own plan development reasoning will effectively translate the parts of the overall service description necessary to locate the needed information in its own knowledge base.

Figure 2 sketches a typical situation where a personal semantic web agent, which we will call 'MyAgent,' is tasked by user Mark to buy a book about XML from an unspecified book sales service on the web. MyAgent knows a number of things about Mark, and it knows how to contact commercial semantic web services. MyAgent first queries a matchmaker to find a suitable service and discovers there is one called Books4Sale, with an OWL-S process model as shown. The Books4Sale process model states that, when correctly invoked, it has the *effect* that the **client** will **own** an **Item** with the specified **title** and **author**. MyAgent can plan to achieve its goal by determining a set of inputs to this service that will produce an effect matching its goal, and then using the accompanying OWL-S service *grounding* to execute the action by formulating a WSDL message that can be passed to the Books4Sale service.

The Books4Sale process model specifies as required inputs an **Item title** and **author**, a **credit card number** and **expiration date**, and a **shipping address**, described using terms from its domain ontologies. It produces as output an order confirmation number and shipping tracking number (not shown). The problem is that MyAgent uses an ontology (MyStuff) different than that used by Books4Sale. MyStuff includes a class **Book** with properties '**by'** for the author and '**name'** for the title that is used to represent the object of its goal. Furthermore, **Items** in the Books4Sale service ontology represent different *sets of books* (identified internally by ISBNs), each with an associated quantity in stock, while instances of the MyStuff class **Book** represent individual books, each assigned an inventory number by the owner when purchased. These differences illustrate the kinds of problems that arise when various developers design ontologies for related but different purposes.

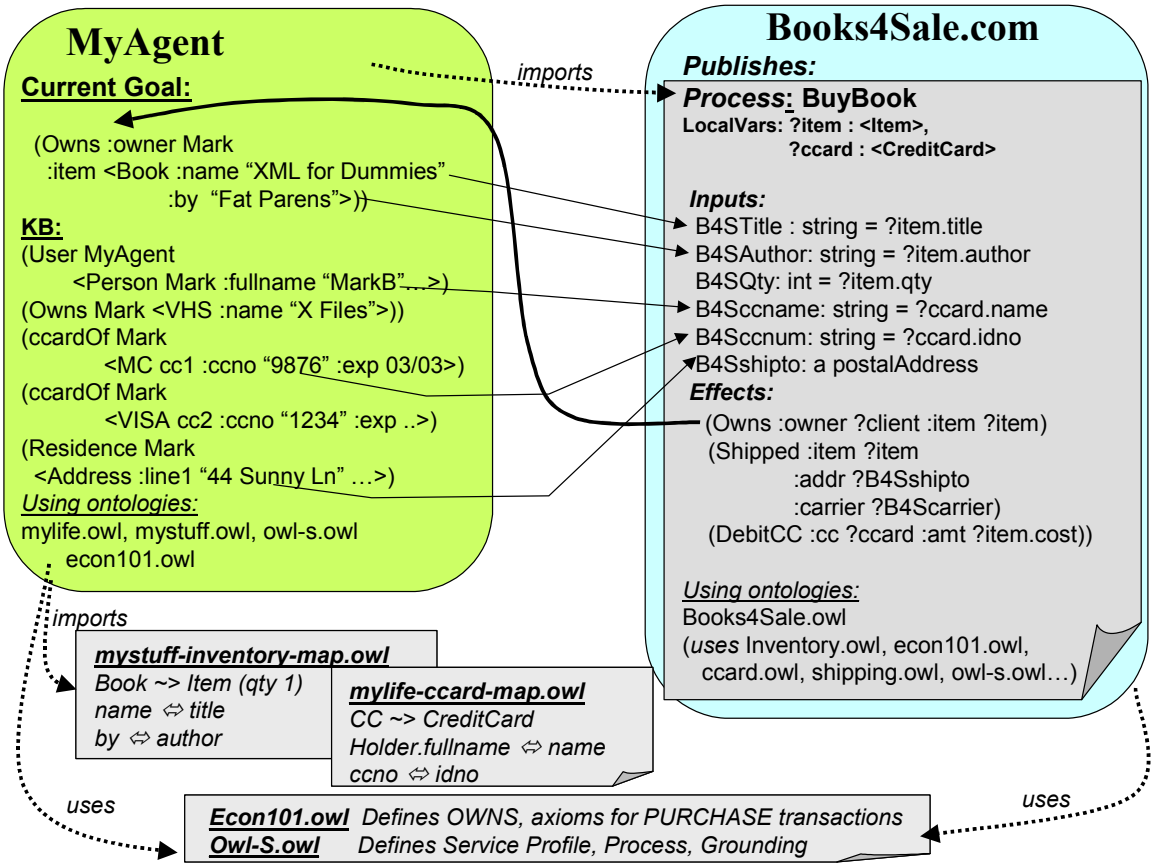

Figure 2: Book Buying Example

MyStuff	*Bridging relation*	Inventory
Book	*InstanceClass-SetClass*	Item
Book.name	*Equivalent Property*	Item.title
Book.by	*Equivalent Property*	Item.author
Book.pubDate	*Equivalent Property*	Item.PubDate
Book.purchDate	- N/A -	
Book.itemno	- N/A -	Item.ISBN
	Cardinality(inst) = 1 >>	Item.qty

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

Table 1 summarizes the partial mappings between terms in the two ontologies. The two concepts can be related *for the purposes of the transaction* as long as the key descriptive properties of the class Book (needed as service inputs) have mappings to the corresponding descriptive properties of Item, and the bridging axioms capture the condition that an Inventory:Item description with qty =1 can refer to the same entity as a description of a MyStuff:Book.

By expressing these mapping rules as additional axioms in the planner's knowledge base, the required inputs to the BuyBook process associated with the client's goal can be identified by the planner during goal regression. 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 conditions 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' pre-conditions must be able to be satisfied by the client. The catch is that this goal regression reasoning will only succeed if the constraints are translated into the client's ontology, either in advance by translating the whole service description or incrementally by the planner by backward chaining in the presence of the necessary ontology-mapping rules.

On the semantic web, *translating* a service process model like BuyBook need not be done by a middle agent if the necessary ontology mappings are available directly. Each client can find and read any ontologies that it does not have which are referenced in messages. If the semantic web architecture were to support mechanisms for finding and loading ontology mappings, then the client could relate descriptions in unfamiliar ontologies to its own knowledge planning processes. Taken together with its own axioms, these mapping rules enable it to complete its reasoning whenever the ontology mappings cover all terms used in a service's parameter type constraints, preconditions and effects.

Figure 3 shows schematically the situation within the client after loading the service description, the published ontologies it references and mappings of terms in those ontologies to those of the client. There are sets of ontologies used internally by the client (O_{Ci}) and by the service (O_{Si}), and some shared ones (O_{Shared}), like OWL-S. There are also published mappings (M) that relate terms in some of the ontologies of the two partners. The client is thus able to extend its reasoning to elements of the service ontology O_{Si} for which mappings are known, reasoning about those parts of the merged ontology as if the client shared them with the server.

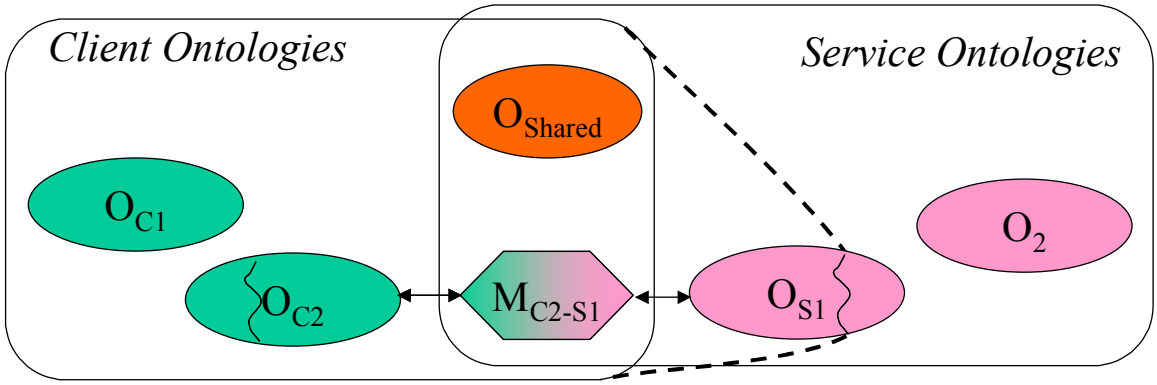

Figure 3: Ontology Merging by loading published ontologies and mappings

Once the client is able to interpret the information requirements of the service, and determine how to satisfy those requirements, it can generate the service request. It may even be possible to construct a valid request with an incomplete set of mappings, if the missing mappings cover optional service inputs. For example, if Books4Sale also used the Item class to represent other goods besides books that it sold, but with required different service parameters to specify those items, then the fact that there were no mappings to the MyStuff ontology to translate those service parameters could be ignored. In contrast, a separate translation agent, called to translate the full service description for the client might fail because no complete mapping between the ontologies was available. Furthermore, when the client is responsible for translation, there is the possibility of getting assistance from the user to identify whether the missing mapping information is relevant, or perhaps even to supply the mapping.

Elsewhere [Dou02, Dou03] we described an implemented approach to translation based on this model that can be used to translate datasets (OWL/RDF files). This system (OntoMerge) works by reading the source and target ontologies into a first-order inference engine (OntoEngine), along with a set of *bridging axioms* or *articulation rules* [Mitr00] that act as translation rules between terms of the two ontologies. The ontologies can be loaded together without clashing because of the distinct URIs used to identify terms in each model, and these URIs are then linked through references in bridging axioms. If the bridging axioms are bi-directional, translation can be performed in either a push or pull fashion. With the push approach, a dataset is translated by asserting its content and forward chaining to find implications expressed in the target ontology. Similar techniques based on applying sets of rewrite rules have been described in [Mitr00, Cha00], although the latter system uses more syntactic transformational rewrite rules. These techniques are useful for translating fully specified messages, such as query or service responses.

With OntoMerge, however, you can also use a 'pull' approach, in which queries are expressed in the target ontology, and OntoMerge backchains to find the answers given the assertions made in the source ontology. We have used this 'pull'

approach in the reasoner of the OPTOP planner to demonstrate the planning of service invocation requests even when the services are described using OWL-S and 'foreign' domain ontologies. As the planner reasons about how to satisfy its own goals (by unifying them with service effects) and infer whether the service's preconditions can be met, its backward chaining naturally utilizes the bridging axioms to locate the information required for service inputs. For example, after loading the bridging axioms of Table 1, OPTOP determines that the title and author inputs to BuyABook can be unified with the 'name' and 'by' fields of the book specified as its internal goal.

Identifying values for non-goal related inputs

The reason direct translation of client goals is not sufficient to request services is that services require inputs that are not directly parts of those goals. Many other pieces of information, such as a login ID and password may be required. If such required inputs are simple facts known to the client, then these facts can be found by the client during precondition evaluation using the technique outlined above, since required inputs can be treated like *knowledge preconditions* of the process. But consider the credit card and shipping information required in our sample book buying process. The knowledge precondition for a shipping address is to know the address to which the purchase should be shipped (a service effect), and this may depends on such things as whether it is intended as a gift or not. Decisions of this kind are common when dealing with unfamiliar web services, although when dealing with commercial services there are a number of common types of required information that can be anticipated even when the specific service to be used is unfamiliar. For example, since purchasing anything requires some form of payment, one can anticipate the need to decide among the client's available credit cards, and implement a decision criteria based on the cost and purpose of the items and the available balances of the cards. Whether to buy a warranty is another common question. Less common, more item specific requirements would include special wrapping preferences, non-standard shipping methods, or whether to buy item-specific optional accessories, etc.

One way to handle the common options on web service requests is to give the client agent default policies for deciding such questions. For example, there could be a default rule that purchases should be shipped to the address of the buyer (user). It is a default rule since it would not apply when a specific location was specified as part of the user's goal. Most current AI planning systems deal with non-goal related effects only to the extent that they try to avoid effects that negatively impact their plans. But many service inputs effectively ask the client to refine the goal or its object (e.g., new or used book?), or require the client to further specify the method by which it is achieved (e.g., selection of a shipping method). Planning systems that can use background goals and preference policies to make these additional distinctions will be needed to more fully automate web service invocation.

These examples highlight why request translation is primarily about making the service description useful to the client rather than translating the request message the client produces once it knows what is required. Since one cannot anticipate all of these input requirements from unfamiliar services, one must translate the constraints on input parameter's implied by service preconditions and effects into knowledge requirements so that the client's decision policies can be applied correctly. Assuming that the basic structure of each service description is provided by the shared OWL-S ontology, then what remains is the domain specific elements of the model. Either the client translates individual input requirements as it plans its request, as we have suggested, or the service description is entirely translated for the client by a translation agent beforehand. One problem is that ultimately the client must also apply the service grounding to send the message, and the grounding was written assuming that the inputs are represented in the service provider's ontology. Thus, after determining a valid set of request inputs, the client must translate these back into the service's language, another step that would have to be done remotely.

Another problem with the external translator approach is that it may fail if there are missing mappings, while in the former case the client could use a partial set of mappings to reason directly with the published description, and then ask for user help to decide whether and how to translate any optional parameters for which mappings were unavailable. The specific distinctions made by a service may lead to the discovery of missing mappings because the service ontology may make distinctions not known to be relevant to the client agent's ontology developer, and mappings cannot exist where one community or the other does not represented the concepts or relations involved. For example, the MyStuff ontology may not represent the distinction between hardcover and softcover books. If that is a required input to a book buying service, the requirement for it would fail to be translated, and user advice would be needed. Ultimately, we will need to develop mixed-initiative approaches to automated service interaction in such open domains that enable our client agents to get additional user assistance, and learn incrementally about the ontologies of unfamiliar services and their mappings into local ontologies.

Translating Responses to Service Requests

Thus far, we have suggested why service invocation may involve translation within a client, rather than using a middle agent. We now ask whether there are circumstances where knowledge local to a particular agent is needed to translate service

responses. Translating responses to requests or queries is perhaps the most well studied kind of translation. Here, the message to be translated is fully formed by one agent (the service provider) independent of the recipient's knowledge state, and the translator must simply identify the mappings needed to transform the description into one in the recipient's ontology. Most of the work on heterogeneous information retrieval is focused on the problem of developing the necessary ontology mappings for this [Rah01].

In most cases, response translation can be done by any agent that has access to the source and target ontologies and the necessary articulation or mapping rules. However, there are some circumstances where the mappings cannot be represented precisely, such as when the classification of particular items depends not on the structure of the relations in the ontologies but on their specific extent. Gio Wiederhold has frequently illustrated this by explaining that different administrative offices at Stanford University use slightly different models for who qualifies for membership in the class 'Employee'. One includes only salaried employees, while the other is broader, and includes non-salaried staff, such as Emeritus Professors like himself. The distinction between these classes is not modeled by systems in either office because neither database represents any attributes that capture the distinction. Thus, no translation rule can be written that succinctly captures the exception to the rule associating the two classes. The only way to translate data about particular individuals being communicated between the two offices is to access the set of all known employees in the target environment and test whether each person whose information is being transmitted is an employee. Effectively, in such situations, the translation needs to be done by the recipient, as a middle agent would not have all the data necessary.

Translation Issues Associated With Dynamic Service Discovery

The final question we consider is the architectural placement of translation functionality for queries to semantic web service matchmakers. Recall that matchmakers receive advertisements called *service profiles* from services that wish to be utilized, and clients looking for services find them by querying matchmakers with general descriptions of their requirements *represented in the form of a partial or abstract service profile.* Pointers to candidate services are returned for consideration by clients.

So what happens when different services use distinct ontologies to advertise their services? If a matchmaker is to maintain a catalog of service advertisements, then a range of possibilities must be considered. At one end of the spectrum, the matchmaker might use its own set of ontologies to organize the services in a single framework. At the other end of the spectrum, it would maintain every service profile in its original form. Call the first approach the Yahoo approach and the second the Google approach., since Yahoo uses a uniform taxonomy constructed by its staff, and Google indexes web pages based on the content of the original sources.

With the Yahoo approach, each service advertisement must utilize concepts in the matchmaker's ontology to describe the represented service. Clients must then translate their queries into abstract service descriptions using the matchmaker's ontology. With OWL-S profiles, this would mean finding concept(s) for the class of service desired (e.g. a Restaurant), the classes of acceptable outputs (Italian food*)*, and desired effect (*e.g.,* Food served in-house vs. packaged for take-out) and quality of service (e.g., max price). You might recognize the problem with this from using an on-line yellow pages. The client needs to be mindful of the expected number of answers. For example, if one wants a to find a Chinese restaurant in Rome, is it better to ask for a service that will provide listings of Chinese Restaurants in Rome, or just restaurants in Rome, or Chinese restaurants in Italy, etc. Finding a useful, targeted answer will depend on the number of candidates in each of those somewhat different conjunctive classes. So an *effective* translation of a service query could depend on both the specific mappings of terms into the matchmaker's ontology and on the selection of *appropriate abstractions* of those terms to find a small, targeted set of answers. If such knowledge is routinely needed to get the best results, then an architecture where the matchmaker does the translation is preferable, since the matchmaker can reformulate the query in a context where knowledge about what parts of the registry space are sparsely populated is available.

With the Google approach, services are allowed to describe themselves using their native ontologies. As a result we see a different set of motivations for having the matchmaker translate queries. Here, it is possible in the extreme case that every new service advertisement is described using a different set of ontologies. How can a client agent translate its own query, or have a middle agent do it when each query must be compared to many profiles, all represented using different ontologies? The only possible strategy requires matching and translation to be performed effectively simultaneously within the matchmaker.

For each kind of semantic service matchmaking, then, there seem to be good reasons to argue that the matchmakers should do the translation of queries they receive. Of course, these are extreme characterizations of the ontology mismatch problems to be faced in connection with service discovery. It is still an open, empirical question how best to design matchmakers that handle a wide diversity of service description ontologies.

Conclusion: The Impact of Knowledge Locality on Translation Processes

An important motivation for the development of semantic web services is their promise of greater and more dynamic interoperability among agents and services. If the semantic web is going to be based on the principle that a thousand ontologies can bloom, then it is important to look at the how translation processes will be used to enable services to interoperate. These questions have not been a major focus in the semantic web services research community to date.

In this paper, we have reviewed the assumptions and motivations for the OWL-S approach to dynamic discovery and utilization of semantic web services, in order to frame a discussion of the multiple roles that ontology translation processes must play in facilitating semantic interoperability. Along the way we suggested why published ontology mappings will be important, and concluded that it is often better to support translation as a function within agents, rather than separate mediators, in semantic web service architectures.

The question of how and where translation processing happens is critical to the design of architectures for semantic web services because of the many related, partially overlapping ontologies in use by different communities on the semantic web. For good reasons, each community wants to control its own ontologies. Our concern is with how to enable interoperability of services beyond tight-knit communities, so remote clients can discover and use these services to fill one-time or occasional needs. Semantic Web Service ontologies make it possible for clients to read published semantic service descriptions and reason with them. Translation of the domain terms in these descriptions will also be needed to support message exchanges.

We have argued that translation functionality can critically depend on knowledge local to particular agents. In particular, we demonstrated how translating a client's goal into a request is best done by the client using published service descriptions and ontology mappings, while translating a client's query to a matchmaker is best done by the matchmaker, because the translation depends on the set of ontologies and profiles known to that agent. The take-away message is that the stronger the need is for a translator to access a localized knowledge context during translation, the more difficult it will be to locate that translation process in an independent middle agent.

Acknowledgements

This writing of this paper was funded under a contract with Yale University to the DARPA DAML program, contract number F30602-00-0600.

References

[TBL01] Berners-Lee, Tim, Hendler, James and Lassila, Ora. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[Bu03b] 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.

[Cha00] 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.

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

[Da02] The DAML Services Coalition (alphabetically Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne and Katia Sycara), "DAML-S: Web Service Description for the Semantic Web", *The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.

[OWL-S] The DAML Services Coalition. OWL-S: http://www.daml.org/services/owl-s/

[OWL] 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.

[Dou02] Dou, Dejing, McDermott, Drew, and Qi, Peishen. Ontology translation by ontology merging and automated reasoning. In *Proc. EKAW Workshop on Ontologies for Multi-Agent Systems*, 2002.

[Dou03] Dou, Dejing, McDermott, Drew, and Qi, Peishen Ontology Translation on the Semantic Web. *Proc. Int'l Conf. on Ontologies, Databases and Applications of Semantics, 2003*.

[Fox03] M. Fox and D. Long. "PDDL2.1: An extension of PDDL for expressing temporal planning domains." *Journal of AI Research*. vol. 20. 2003. pp. 61-124.

[Gai03] Gaio, S., Lopes, A., Botelho, L. "From DAML-S to Executable Code", In *Burg, et al. editors, Agentcities: Challenges in Open Agent Environments*, pages 25-31. Springer-Verlag, 2003.

[Kal03] Kalfoglou, Y, and Schorlemmer, M, "Ontology mapping: the state of the art" *Knowledge Engineering Review* 18:1, pp. 1-31, 2003.

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

[McD02a] 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.

[McD02b] McDermott, Drew "Estimated-Regression Planning for Interactions with Web Services" In *Proceedings of AI Planning Systems Conference*, 2002

[McD03] McDermott, Drew "Reasoning about autonomous processes in an estimated-regression planner." *Proc. Int'l. Conf. on Automated Planning and Scheduling 2003*

[McIl02] 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)*, Toulouse, France, April, 2002.

[Mad02] Madhavan, J., Bernstein, P., Domingos, P., Halevy, A. "Representing and Reasoning about Mappings between Domain Models" In *Proceedings of the* Eighteenth national conference on Artificial intelligence, Edmonton, Alberta, Canada . pp. 80–86. 2002.

[Mitr00] Mitra, P., Wiederhold, G. and Kersten, M. "A Graph-Oriented Model for Articulation of Ontology Interdependencies" In *Proc. Conference on Extending Database Technology 2000* (EDBT'2000). 2000. Konstanz, Germany

[Noy02] Noy, Natalya F. and Musen, Mark A. "Evaluating Ontology Mapping Tools: Requirements and Experience" SMI Report, 2002.

[Pao02a] 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.

[Pao02b] Paolucci, M., Kawamura, T, Payne, T.R. and Sycara, K.. Importing the semantic web in UDDI. In *Proceedings of E-Services and the Semantic Web Workshop*, 2002.

[Pao03] Paolucci, M., Ankolekar, A., Srinivasan N.and Sycara, K. "The DAML-S Virtual Machine", In *Proceedings of the Second International Semantic Web Conference* (ISWC), 2003, Sandial Island, Fl, USA, October 2003, pp 290-305.

[Rah01] Rahm, E., and Bernstein, P. A. 2001. A survey on approaches to automatic schema matching. VLDB Journal 10(4).

[Sab03] Sabou, M., Richards, D. and van Splunter, S. “An experience report on using DAML-S,” In *The Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW '03)*. Budapest, 2003.

[Sch00] Schlenoff, C., Gruninger M., Tissot, F., Valois, J., Lubell, J., Lee, J., "*The Process Specification Language (PSL): Overview and Version 1.0 Specification*," NISTIR 6459, National Institute of Standards and Technology, Gaithersburg, MD 2000.

[She03] Sheshagiri, M., desJardins, M., Finin, T. A planner for composing services described in DAML-S,” In *AAMAS Workshop on Web Services and Agent-Based Engineering,* 2003.

[Sir04] Sirin, E., Parsia, B., and Hendler, J. Composition-driven filtering and selection of semantic web services. In *AAAI Spring Symposium on Semantic Web Services*, 2004.

[UDDI] UDDI. The UDDI Technical White Paper. http://www.uddi.org/, 2000.

[Usch02] Uschold, M. and Gruninger, M. “Creating Semantically Integrated Communitities on the World Wide Web” Semantic Web Workshop, Honolulu, May 2002. http://semanticweb2002.aifb.uni-karlsruhe.de/USCHOLD-Hawaii-InvitedTalk2002.pdf.

[Wu03] 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.

[We99] Weld, Dan. Recent Advances in AI Planning. *AI Magazine, 1999.*