Semantic Web Service Offer Discovery for E-commerce*

Jacek Kopecký
Semantic Technology Institute (STI)
Innsbruck, Austria
jacek.kopecky@sti2.at

Elena Simperl
Semantic Technology Institute (STI)
Innsbruck, Austria
elena.simperl@sti2.at

ABSTRACT
Semantic Web Services (SWS) are an important part of the Semantic Web, traditionally focused on discovery and composition of e-services. In the area of e-commerce services, it is necessary to go past the granularity of service discovery and also to consider discovering the actual offers provided by a service. Nevertheless, Semantic Web Services research has only recently started to consider offer discovery. In this paper, we present a solution for offer discovery that uses WSMO-Lite, the new lightweight semantic Web service annotation framework.

Categories and Subject Descriptors
H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*

General Terms
Algorithms, E-commerce

Keywords
Offer Discovery, Web Services, Semantic Web Services

1. INTRODUCTION
For a decade, the World-Wide Web has been a growing market place that makes it easier for companies to reach potential clients with their products and services. Online commerce through web sites and search engines has a much lower cost, compared to the costs of commerce in the traditional channels, such as brick-and-mortar stores and offices, complemented with heavy advertising. In addition to reaching many more potential clients, e-commerce companies can also profit from the long tail or less-popular products.

The Semantic Web is an extension of the current Web with semantic descriptions of data and services that can be used automatically by the computer on behalf of its user. A major technology for publishing services on the Web is the so-called *Web services* (or WS-*) family of technologies. Based on the WWW standards HTTP and XML, Web services are gaining significant adoption in areas of application integration, wide-scale distributed computing, and business-to-business cooperation. Still, many tasks commonly performed in service-oriented systems remain manual (performed by a human operator), and services in areas such as e-commerce mostly remain available only through a human-facing interface (HTML).

In order to make Web services part of the Semantic Web, the research area of Semantic Web Services (SWS) aims to increase the level of automation around Web services. SWS automation is supported by machine-processible semantic descriptions which capture the important aspects of the meaning of service operations and messages. SWS descriptions are processed by a semantic execution environment (SEE, for instance WSMX [3]). A user can submit a concrete *goal* to the SEE, which then accomplishes it by finding and using the appropriate available Web services. SWS research focuses mainly on how the SEE "finds the appropriate Web service(s)", as illustrated in Figure 1 with the first four SEE tasks.

In the figure, the user wants to arrange a June vacation in Rome. There are four services with published descriptions: the airline Lufthansa, and hotel reservation services for New York, Rome, and one for the Marriott hotel chain worldwide. The SEE first *discovers services* that may have hotels in Rome, discarding Lufthansa which does not provide hotels, and the New York service which does not cover Rome. Then the SEE *discovers offers* by interaction with the discovered services. The available offers in this particular example are only three hotels: a 4* Marriott at the outskirts of Rome, and one 2* and one 3* hotel in the city center. Then the SEE *filters* the offers depending on the user's constraints and requirements (minimum 3-star rating), *ranks* them according to the user's preferences (central location is more important than price) and *selects* one offer, in the end *invoking* the corresponding service.

For simplicity of the illustration, in this figure we omit *service composition*, which combines multiple services in order to achieve more complex functionalities; and *semantic mediation* which resolves any data and process heterogeneities.

*This work is partially funded by the EU research project SOA4All.

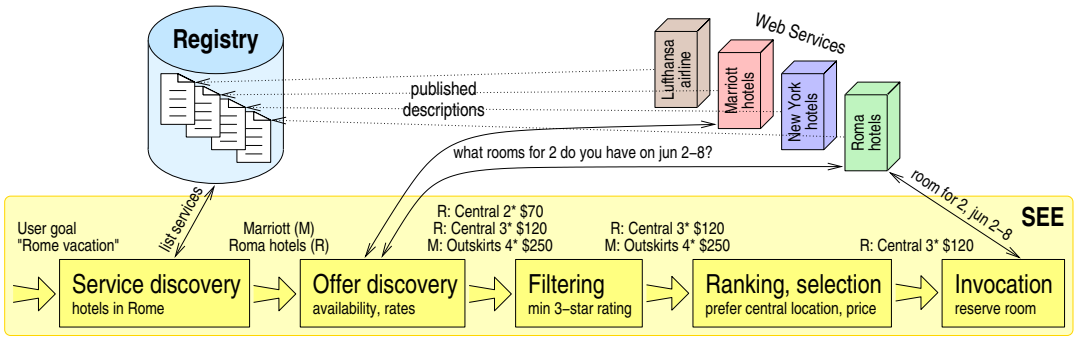

Figure 1: Semantic Execution Environment (SEE) automation tasks

There are a number of SWS frameworks that support SWS automation; the major two are WSMO [9] and OWL-S [10]. Both are built from the top down, providing semantic descriptions tailored for the expected automation tasks. However, neither of these frameworks supports the particular task of *offer discovery,* detailed in [6] and necessary especially for e-commerce. To add offer discovery, these frameworks have to be extended with further constructs, such as a "data-fetching interface" introduced in [12].

Recently, we have proposed WSMO-Lite [11] as a lightweight SWS framework that adds semantic annotations as a layer on top of established Web service technologies. It is built from the bottom up and it provides semantic annotations for WSDL[1], based in a clean model of service semantics, instead of tailoring the semantic constructs towards any particular tasks. As a consequence, WSMO-Lite annotations are more reusable and they make it easier to realize various SWS automation functionalities. In this paper, we show **a concrete realization of offer discovery** which uses WSMO-Lite semantic descriptions.

We should note that what we call *offer discovery* is elsewhere in literature (e.g. [4]) called *service discovery*, making the distinction between a *Web service* and the *service* it actually provides. We prefer the term *offer* to avoid causing confusion due to overloading of the common word service. In addition, the term offer is easily understood to encompass both offered services and offered products, the basis of e-commerce.

As a final note, there are online services that aggregate data from many providers and make it possible for users to compare the offers from different sources. Such services can certainly be created without the use of semantics, by tailoring a data aggregation connector for every participating provider, or by standardizing a common offer discovery interface. Even the standardized interfaces would likely be different in disparate domains, making cross-domain aggregation resort to specific connectors again. An example of such a service is `expedia.com`, which puts together hotel reservations, plane tickets and car rentals. In Semantic Web Services, and in particular in our work, we propose the use of semantic technologies to address this common and well known integration problem. Our offer discovery mechanism uses light-weight semantic annotations to be able to access any suitable Web service, regardless of the domain of its offerings.

This paper is structured as follows: in Section 2, we define offer discovery. In Section 3, we discuss WSMO-Lite and the semantic annotations necessary for offer discovery, and Section 4 provides details of our implemented offer discovery algorithm. Finally, Section 5 lists some related work, and Section 6 concludes the paper.

2. WEB SERVICE OFFER DISCOVERY

Within the big picture of SWS automation, offer discovery follows the task of service discovery, and its results go into filtering, ranking and selection. Service discovery returns a set of services that can potentially fulfill the user's goal. Offer discovery interacts with these services (or the service providers) in order to find out any concrete offers that are relevant to the goal; the result of offer discovery is the set of available offers. This set is then subject to filtering and ranking according to the user's constraints and preferences, in order to select the best offer. In the end, the selected offer is *consumed*, i.e. the client invokes the service that gave this offer and, after successful invocation, it will get the offered product or functionality.

In order to be able to talk about offer discovery, we need to specify what we mean by the term "offer". From the point of view of contracting, an offer is a contract proposed by the service provider to the client, who can evaluate and accept or reject it. Naturally, an offer can only be valid for a limited period of time; this consideration, however, is not yet included in our work.

With Web services, there is generally no specific interface that would explicitly talk about contracts or offers and their acceptance or rejection by the client. Instead, a Web service may provide a set of *information inquiry operations* that may return information about what the service offers, e.g. finding out hotel room availabilities for given dates. The client may reject the offer by simply ignoring that data, and it may accept it by calling *execution operations*, i.e., the operations that invoke the actual functionality of the service; e.g. booking a room in a given hotel for the given dates.

Therefore, for Web service offer discovery, we can formalize an offer O as a tuple that contains two sets of parameters: execution parameters P_x that are required for invoking the service and consuming this offer, and extra parameters P_e

[1]Web Service Description Language, `http://w3.org/TR/wsdl20`

that help the client to filter and to rank the offers:

$$O = \langle P_x, P_e \rangle$$

In our hotel scenario, the execution parameters are the start and end dates of the stay, the number of guests and any required data about them; and of course the selected hotel reservation service and the concrete hotel. All this data is necessary for making a reservation. The extra parameters for filtering and ranking would be the locations and star ratings of the hotels and the room prices. Strictly speaking, the client does not need to know the price when reserving a room, and it certainly does not need to send the price to the service in the process of making a reservation.

Some of the execution parameter values come from the user goal, in our case the data of the guests and the dates of the stay. These parameters will be the same for all the available offers, so they cannot serve any purpose during the filtering and ranking stages; however, they will be necessary for invoking the service. In fact, the particular service that supplies a given offer is itself an execution parameter of the offer. This ensures that the offer is a self-contained construct for the further SWS automation steps: the invocation component knows what service to invoke; and in filtering and ranking the client may express constraints or preferences directly on the services, for instance by building trust with particular providers that delivered good value in the past.

Any other offer parameter values come from the offer discovery process, interacting with the information inquiry operations of the discovered services.

The split of offer parameters into execution parameters and extra ones allows us to establish identity for offers — two offers are equivalent iff their execution parameter sets are the same:

$$O_1 = \langle P_x^1, P_e^1 \rangle, O_2 = \langle P_x^2, P_e^2 \rangle : \; O_1 \equiv O_2 \Leftrightarrow P_x^1 = P_x^2 \quad (1)$$

This equivalence relation comes from the fact that only the execution parameters are used in the invocation phase, when an offer is consumed. If we had two offers that would vary only in the extra parameters, the service could not know which of the two offers the client intends to consume. Hence, the offer discovery process must assure that it does not produce different but equivalent offers.

To finish the formalization, offer discovery is a function (we call it *DiscO* below) which maps a set of discovered Web services $\{S_i\}$ into a set of non-equivalent (Eq. 4) offers $\{O_j\}$ from these services (Eq. 5):

$$DiscO(\{S_i\}) \rightarrow \bigcup_{s \in \{S_i\}} DiscO(s) \quad (2)$$

$$DiscO(s) \rightarrow \{O_j^s\} \quad (3)$$

$$\forall o_1, o_2 \in \{O_j\} : \; o_1 \equiv o_2 \Leftrightarrow o_1 = o_2 \quad (4)$$

$$\forall o \in DiscO(s), o = \langle P_x, P_e \rangle : \; s \in P_x \quad (5)$$

While we define offer discovery to deal with a single discovered service at a time (cf. Eq. 2 and Eq. 3), it is possible that a more sophisticated offer discovery approach can negotiate with multiple services in parallel and pitch them one against another in order to get better deals. For example, a retailer can promise to match any competitor's price, so the offer discovery process would need to get the competitors' offers first and then use them to get matching counteroffers from the retailer. Even though this kind of multi-way negotiation is sometimes possible in the real world, we have not seen a single example of an e-service with such capabilities, therefore we leave this as an extension to be revisited in the future.

3. WSMO-LITE SEMANTIC SERVICE ANNOTATIONS FOR OFFER DISCOVERY

Semantic offer discovery should optimally be able to communicate with any Web service that provides operations for finding information about its offers. To achieve this, the offer discovery engine needs a description of the service interface, to see what operations it contains that can be used to gather offer information; and a description of the exchanged data, to understand the offers and to be able to compare them against the goal. We use the WSMO-Lite SWS description framework [11], together with the underlying SAWSDL standard[2], to provide all the necessary semantic descriptions.

Any Web service, described in WSDL, has an *interface* which consists of a number of *operations*. WSMO-Lite annotates the Web service interfaces and operations with *functional semantics*, and the inputs and outputs of the operations are annotated with *information semantics*. Functional semantics are expressed either using functionality classifications (such as the service classifications from the ecl@ss taxonomy[3]), or using logical pre-conditions and effects of the service or its individual operations. Information semantics are expressed in WSMO-Lite using ontologies.

Web service interfaces often intermix operations for offer inquiry with operations that actually provide the resulting product or service, for instance a hotel reservation service would provide the availability inquiry operations along with the operations for making reservations. For the purposes of automated offer discovery, we will use operations that only provide information and do not have any significant side-effects. In other words, we need what the Web architecture [1] calls "safe interactions". Information retrieval is the canonical example of a safe interaction: the client may query a service about the availability of hotel rooms, yet by issuing the query the client makes no commitment to book the room.

To indicate operation safety, WSDL 2.0 defines an extension attribute `wsdlx:safe`[4]. Figure 3 shows a sample hotel booking interface with a safe availability inquiry operation (`findAvailableHotels`), a safe price inquiry operation (`getRoomPrice`), and the operations for making or cancelling

[2]Semantic Annotations for WSDL and XML Schema, `http://w3.org/TR/sawsdl`

[3]eCl@ss Standardized Material and Service Classification, `http://eclass-online.com/`

[4]`http://www.w3.org/TR/wsdl20-adjuncts/#safety`

```
<wsdl:types>
  <xs:schema targetNamespace="&hotels;">
    <xs:element name="availabilityQuery"
        sawsdl:modelReference="&onto;StartDate
                               &onto;EndDate
                               &onto;GuestInfo">
      ... content dropped for brevity ...
    </xs:element>
    ... further elements dropped for brevity ...
  </xs:schema>
</wsdl:types>

<wsdl:interface name="HotelReservation">
  <wsdl:operation name="findAvailableHotels"
                  wsdlx:safe="true">
    <wsdl:input element="availabilityQuery"/>
    <wsdl:output element="availabilityResponse"/>
  </wsdl:operation>

  <wsdl:operation name="getRoomPrice"
                  wsdlx:safe="true">
    <wsdl:input element="roomPriceQuery"/>
    <wsdl:output element="roomPriceResponse"/>
  </wsdl:operation>

  <wsdl:operation name="reserveRoom">
    <wsdl:input element="reservationRequest"/>
    <wsdl:output element="reservationConfirmation"/>
  </wsdl:operation>

  <wsdl:operation name="cancelReservation">
    <wsdl:input element="cancellationRequest"/>
    <wsdl:output element="cancellationConfirmation"/>
  </wsdl:operation>
</wsdl:interface>
```

Figure 2: An excerpt from a hotel reservation service WSDL description with WSMO-Lite annotations, incl. WSDL 2.0 operation safety

a reservation (`reserveRoom`, `cancelReservation`), both of which are by definition not safe. The attribute `wsdlx:safe` is equivalent to a SAWSDL *modelReference* annotation that has the value set to `http://www.w3.org/ns/wsdl-extensions#SafeInteraction`. This annotation is then treated as a WSMO-Lite functionality category for safe operations. It remains to be seen whether all safe operations can be treated as offer-inquiry operations, but due to their safety, there is no harm in invoking such operations even if they do not actually help get information about the service offers.

Further, as also shown in the figure, WSMO-Lite annotates operation inputs and outputs with pointers to ontology entities, such as RDFS classes. These annotations allow the semantic client to match its goal data against the inputs of the offer-inquiry operations, and to match the goal and offer data against the inputs of the execution operations, to distinguish between execution and extra parameters.

In summary, WSMO-Lite provides sufficient, albeit lightweight, semantic descriptions for us to realize an automated offer discovery process, described in the following section. Note that none of the annotations are specific to offer discovery; both the operation safety flag and input/output annotations are useful for other purposes.

Inputs: Service S whose offers should be discovered, user goal G.
Result: The set of offers provided by S.

```
1  set up initialOffer with exec. parameter values from G, S,
2        incompleteOffers = {initialOffer},
3        completeOffers = {}
4  identify offerInquiryOperations from S
5  while incompleteOffers is not empty
6      offer = first(incompleteOffers)
7      if offer is complete (all exec. parameters are present) then
8          move offer to completeOffers; continue while
9      set up knowledgeBase from offer and G
10     select neededOfferExecutionParams (not satisfied by offer)
11     plan = planOperations(
12                 initial state: knowledgeBase,
13                 goal state: neededOfferExecutionParams,
14                 operations: offerInquiryOperations)
15     if empty(plan) then eliminate offer; continue while
16     receivedValues = invoke(S, plan[1], knowledgeBase)
17     add receivedValues to offer (possibly multiple offers)
18 end while
19 gatherExtraParameters(completeOffers)
20 return completeOffers
```

Figure 3: Offer discovery algorithm

4. OFFER DISCOVERY ALGORITHM AND IMPLEMENTATION

The algorithm shown in Figure 3 shows how we have realized offer discovery on top of the semantic annotations described above.

In short, the algorithm starts by creating an initial offer from the goal data, then it keeps invoking the offer-inquiry operations for every offer that does not provide all the required execution parameters (so-called incomplete offers).

To get from the goal data to all the required execution parameters for each offer, we use *AI planning* (cf. [8], line 11), taking the currently known data (from the goal and from the current offer) as the initial state, the presence of all execution parameters as the goal state, and the offer-inquiry operations as the possible transitions. In our hotel scenario, booking a room requires the dates and guest data (from the user goal) and a selected hotel, so the algorithm would invoke an operation such as `findAvailableHotels(dates,guests)`, and the resulting list of hotels would complete the list of offers. This shows that the data received from a single operation can imply creating multiple offers from the current one (line 17): the initial incomplete offer only had the dates and guest data filled in and not the hotel, and `findAvailableHotels` returns multiple hotels for the same dates and guests.

In the end (line 19), the algorithm tries to gather further extra parameters for all the known offers. This is necessary because the main algorithm only satisfies the execution parameters, so it would never invoke an operation such as `getRoomPrice(hotel,dates)`, because the resulting *price* value is not an execution parameter. We are working on heuristic approaches for selecting the operations to invoke for the extra parameters.

The execution parameters mentioned on line 1 are the input parameters of the execution operations. While all non-safe

operations can be treated as the execution operations, it is necessary to select only those execution operations that are relevant for the user goal: for instance, the hotel reservation service has a `cancelReservation` operation which is not going to be invoked when booking a room, and whose input parameter (a reservation code) cannot be satisfied by the goal or any discovered offers.

We have implemented the algorithm within the WSMX system [3] with positive initial results, but a larger evaluation experiment remains as future work. Since WSMO-Lite does not contain any mechanism for describing user goals, we use goal description captured in WSMO, as is done in the whole of WSMX.

Currently, the planning algorithm used in our implementation is not semantic (for instance, it cannot take into account any subclass relationships between the parameters and the available values); this limits the expressivity available for the ontologies used to describe the input and output data of the service operations. We are looking for suitable ways of extending the planning part with semantic reasoning.

5. RELATED WORK

There are two categories of related work: earlier research applicable to solving the offer discovery problem, discussed in Section 5.1, and other proposed solutions for offer discovery, enumerated in Section 5.2.

5.1 Applicable earlier research

Semantic Web service offer discovery, as defined in the preceding sections, is related to earlier research in automated negotiation and contracting, and also to the area of information gathering.

The term *negotiation* has been used for different purposes in a variety of computer science fields, e.g. e-commerce, grid computing, distributed artificial intelligence and multi-agent systems. In e-commerce, Beam and Segev [2] define negotiation as "the process by which two or more parties multilaterally bargain resources for mutual intended gain". There are several different types of negotiations in e-commerce: auctions (multiple buyers bid for price), double auctions (both buyers and sellers bid for price, e.g. stock exchanges), one-to-one bargaining, and even catalogue provision (price fixed by seller). Offer discovery is similar to catalogue provision (offer discovery accesses and retrieves the relevant parts of the offer catalogue), but it could be extended in the direction of bargaining as well.

Research in information gathering has dealt, among other issues, with using multiple information sources to gather the requested (or relevant) information (cf. [5]), based on a user query. In Semantic Web services, a user goal can be seen as a form of query, and the discovered Web services (or their individual operations) as information sources. For SWS offer discovery, we could potentially shape the description of the services and their operations so that information gathering techniques would be applicable. This remains to be investigated.

We can see that offer discovery is not a problem specific to SWS. However, earlier efforts on similar automation (e.g. in multi-agent systems) have generally presumed a controlled environment with a predefined set of interaction protocols for various tasks; for instance, a marketplace would dictate a bargaining and auctioning protocol. Such an approach can be applied to Web services, however, a bargaining/auctioning protocol or a common query language would need to be standardized and adopted by most service providers. Any SWS offer discovery mechanism, together with the necessary semantic annotations mechanisms, would be different and novel because SWS offer discovery aims to be generic, independent of the domain of the service offers. Indeed, the semantic annotations should make the offer discovery algorithm adapt to any available negotiation or query protocol.

5.2 Alternative proposed SWS offer discovery approaches

At the time of this writing, we know of two published attempts that involve dynamic offer discovery in Semantic Web Services: the use of a "contracting interface" by Zaremba *et al.* [12, 13] and an "estimation phase" of discovery by Küster *et al.* [7].

Zaremba *et al.* talk about a so-called "contracting interface" with a described operation choreography. In Zaremba's contracting phase of Web service discovery, the SEE client follows the predefined choreography to get information about the relevant offers from a discovered Web service. The contracting interface can be likened to a prescribed protocol for offer discovery. While Zaremba's approach is workable, there are difficulties with limiting the amount of information that will be retrieved from the service, as this complicates the design of the contracting interface's choreography. In comparison, our approach needs no explicit choreography, which is replaced by AI planning and heuristics; both Zaremba's and our approaches require the annotations of operation inputs and outputs.

Küster *et al.* describe a service basically as a template for the offers, and parts of the offer datastructure are marked as "estimation phase" parameters, with simple ordering of predefined interactions by which the client can retrieve the relevant offer data. This approach seems to require that the user's goal and the service description have a very similar structure, which may require complex mediation in heterogeneous environments. Our approach does not require that the structures of service and goal descriptions are similar in any way, instead it only deals with the exchanged data; in data mediation, the requirements of our approach are the same as those of the two presented alternative approaches.

We closely follow the developments of these approaches to offer discovery, and we intend to perform a larger evaluation of our solution in comparison against these alternatives.

6. CONCLUSIONS

Web services are a necessary part of the Semantic Web, and research on Semantic Web Services aims to automate their use. Among the tasks that can be automated using semantic technologies is offer discovery, especially important for e-commerce applications. In this paper, we have presented a formalization of offer discovery and we have shown an offer discovery algorithm and its implementation using light-

weight WSMO-Lite semantic descriptions.

The main benefit of using semantics is that a semantic offer discovery mechanism need not really understand the semantics of any specific web service interfaces (apart from the inputs and outputs). In contrast, traditional e-commerce data aggregation applications such as `expedia.com` need to have special code for any partner interface with which they interact, which has negative impact on the costs of maintenance and evolution of the system.

While our prototype gives positive results, there are some open points in need of solutions, and we need to perform a proper evaluation experiment for our approach.

7. REFERENCES

[1] Architecture of the World Wide Web. Recommendation, W3C, December 2004. Available at `http://www.w3.org/TR/webarch/`.

[2] C. Beam and A. Segev. Automated negotiations: A survey of the state of the art. *Wirtschaftsinformatik*, 39(3):263–268, 1997.

[3] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler. WSMX – A Semantic Service-Oriented Architecture. *International Conference on Web Services (ICWS 2005)*, July 2005.

[4] U. Keller, R. Lara, H. Lausen, and D. Fensel. Semantic Web Service Discovery in the WSMO Framework. In J. Cardoses, editor, *Semantic Web: Theory, Tools and Applications*. Idea Publishing Group, 2006.

[5] C. A. Knoblock. Planning, executing, sensing, and replanning for information gathering. In *Proc. of the 14th Int'l Joint Conference on Artificial Intelligence*, pages 1686–1693, 1995.

[6] J. Kopecký, E. Simperl, and D. Fensel. Semantic Web Service Offer Discovery. In *Proceedings of Service Matchmaking and Resource Retrieval in the Semantic Web Workshop, colocated with* 6^{th} *ISWC*, 2007.

[7] U. Küster and B. König-Ries. Supporting dynamics in service descriptions — the key to automatic service usage. In *Proceedings of the Fifth International Conference on Service Oriented Computing (ICSOC07)*, Vienna, Austria, September 2007.

[8] D. Nau, M. Ghallab, and P. Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[9] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005.

[10] The OWL Services Coalition. OWL-S 1.1 Release. Available at `http://www.daml.org/services/owl-s/1.1/`, November 2004.

[11] T. Vitvar, J. Kopecký, J. Viskova, and D. Fensel. WSMO-Lite Annotations for Web Services. In *Proceedings of the 5th European Semantic Web Conference (ESWC)*, Tenerife, Spain, 2008.

[12] T. Vitvar, M. Zaremba, and M. Moran. Dynamic service discovery through meta-interactions with service providers. In E. Franconi, M. Kifer, and W. May, editors, *ESWC*, volume 4519 of *Lecture Notes in Computer Science*, pages 84–98. Springer, 2007.

[13] M. Zaremba, T. Vitvar, M. Moran, and T. Hasselwanter. WSMX Discovery for SWS Challenge. SWS Challenge Workshop, Athens, Georgia, USA, November 2006.