Semantic Web Service Offer Discovery

Jacek Kopecký, Elena Simperl, and Dieter Fensel

Digital Enterprise Research Institute (DERI)
Innsbruck, Austria
firstname.lastname@deri.at

Abstract. Semantic Web Services are a research effort to automate the usage of Web services, a necessary component for the Semantic Web. Traditionally, Web service discovery depends on detailed formal semantic descriptions of available services. Since a complete detailed service description is not always feasible, the client software cannot select the best service offer for a given user goal only by using the static service descriptions. Therefore the client needs to interact automatically with the discovered Web services to find information about the available concrete offers, after which it can select the best offer that will fulfill the user's goal. This paper shows when and why complete semantic description is unfeasible, it defines the role and position of offer discovery, and it suggests how it can be implemented and evaluated.

1 Introduction

The Semantic Web is not only an extension of the current Web with more semantic descriptions of data; it also needs to integrate 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. Based on 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).

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 of some of these tasks, e.g. discovering the available services and composing them to provide more complex functionalities. SWS automation is supported by machine-processible semantic Web service descriptions. Current state of the art in non-semantic service description is WSDL[1], which can describe the messages accepted and produced by a Web service, and the simple message exchanges (called operations) and all the necessary networking details. In effect, WSDL specifies a limited syntactical contract that the service adheres to. Semantic descriptions capture the important aspects of the meaning of operations and messages, generally in a formal language based on logics.

SWS descriptions are processed by a semantic execution environment (SEE, for instance WSMX [5]). A user can submit a concrete *goal* to the SEE, which then finds

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

and uses the appropriate Web services to accomplish the goal. SWS research focuses mainly on how the SEE "finds the appropriate Web service(s)", as illustrated in Figure 1 with the first three SEE tasks.

In the figure, meant to be illustrative of the situation, rather than a real-world scenario, the user decides to lead a healthier life and wants to buy 2kg of fruit. The SEE will first discover any services that sell fruit (discarding the service that sells potatoes), then it will filter depending on the user's constraints and requirements (the user doesn't like peeling oranges), ranks the resulting services according to the user's preferences (the user is a student and so prefers the cheaper options) and selects the one service that is invoked in the end. At any stage in the process, the user can be allowed to confirm the results.

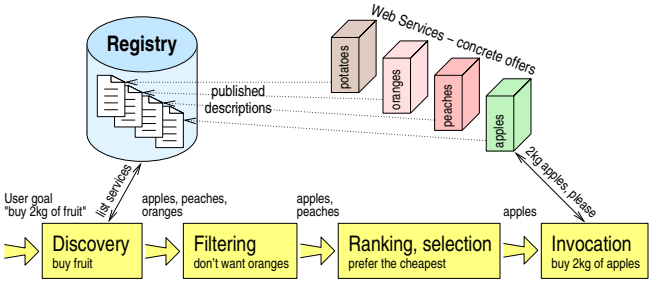

Fig. 1. Semantic Web Services automation tasks

We've chosen the simple fruit-shops example here to illustrate the situation in easily accessible terms, however, the general situation applies to any kinds of service: the existing services will publish their descriptions in a registry; then the SEE will *discover* the services applicable to the goal, *filter* and *rank* them according to any constraints and preferences specified by the user, and *invoke* the selected service to achieve the user's goal.

The steps described above rely solely on the published Web service descriptions to find the best service that matches the user's goal. Alas, in many cases it is not feasible to put all the relevant information in the service description, due to reasons detailed later in this paper. For instance, a grocery shop service would not list all the kinds of fruit they currently sell along with their up-to-date prices; instead, such a service would be described as "selling groceries". This limits the scope of discovery based on static descriptions and introduces the need for an additional step, where the SEE will contact the discovered Web services (or their providers) to find out more about the service's concrete offerings.

This additional step is called **offer discovery** (as opposed to Web service discovery). The objective of this step is to establish whether the discovered Web service can fulfill the user's concrete goal and under what conditions. In our fruit shopping example, the SEE checks whether a grocery shop service carries any fruits, what sorts of fruits are available and at what prices, as shown in Figure 2. In this paper, we detail when and why

static Web service discovery is not sufficient, we describe in detail what offer discovery needs to accomplish and how offer discovery approaches should be evaluated. We do not present a complete offer discovery solution in this paper, as our work is in an early stage.

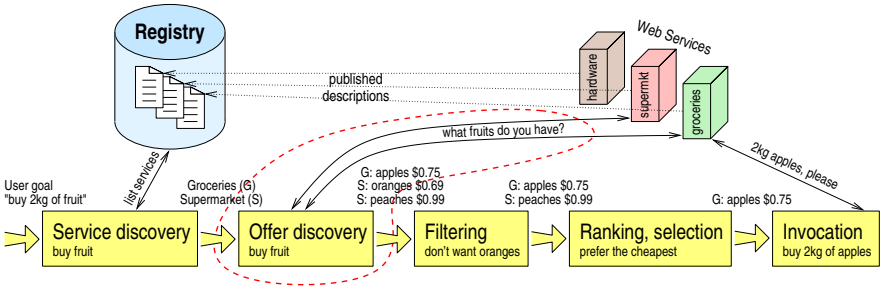

Fig. 2. Semantic Web Service offer discovery in SEE tasks

This paper is structured as follows: in Section 2 we detail the scenarios where offer discovery is necessary. Section 3 defines SWS offer discovery and relates it to other SEE tasks. Section 4 presents related work, both within Web services and in earlier research areas. In Section 5, we sketch the envisioned solution. Section 6 describes our expected evaluation methodology, and Section 7 contains concluding remarks.

2 Limitations of static Web service discovery

The best way to define service offer discovery is by describing the problems that it aims to solve. First, let us review the distinguishable functions of a semantic execution environment (SEE). The following steps are traditionally executed after a user submits their goal "buy 2kg of fruit", as shown in Figure 1.

1. **Web service discovery**[2] — using published descriptions, find all the available Web services that may sell fruits (the services may be more generic, like a supermarket with all kinds of products, or more specific, like an owner of a cherry-tree orchard, who naturally only offers cherries).
2. **Filtering** — filter out services that do not fit the user's constraints (for instance a service that sells oranges, because the user does not like them).
3. **Ranking, selection** — rank the remaining offers based on the user's preferences, for instance by price. The best-ranked service may be automatically selected, or the ranking may be presented to the user.
4. **Invocation** — use the selected service to achieve the goal (in our case, purchase the fruit).

[2] Sometimes, the term *discovery* is used to mean all the steps leading from a user's goal to a service that can fulfill it, i.e. everything but invocation. We choose a narrower definition of discovery which only does matchmaking on the available service descriptons.

There are also the additional steps of *mediation* and *service composition*, but they are not particularly relevant to offer discovery, even though they may interact with it.

The task sequence above is fully adequate when the service descriptions carry all the data relevant for the goal of the user. In a grid environment, a user might need processing services and storage services, and the descriptions will contain such classifications. In our fruit-buying scenario, the services need to advertise in their descriptions the particular kinds of fruit they sell and at their prices (for ranking).

A vast majority of currently available public Web services[3] provides only a limited and fixed number of offers (products, services): a fax service from `oneoutbox.com` has a single operation `SendFax`; the Amazon S3 service at `amazonaws.com` provides data storage and retrieval (two offers); or the typical stock quote services provide one offer, the current (delayed) price of any given stock. As they are described, the fax service works globally for any fax number, the S3 service works with any size and kind of data, and many stock quote services purport to know all stocks, therefore there are no discoverable limitations. The offers of these services are simple and generic.

On the other hand, there are services whose discoverable offers are of a finer granularity. The Amazon E-Commerce Service, for instance, gives access to all products sold on Amazon.com. Since Amazon cannot claim to sell all book and DVD titles, for instance, each book and DVD title and any other product becomes a separate offer. The service has operations for checking the availability or price of any given book title etc. On a similar note, a broadband internet provider only serves certain areas, and it provides operations to check availability at any given address.

Figure 3 illustrates how different kinds of services have widely different numbers of offers[4]. From the left side, a temperature conversion service has two operations for converting either way; a telephony service can have calling, voice mail, sms and a few other operations; a currency exchange service can recognize tens or hundreds of currencies; and on the higher end a hotel reservation service can offer tens of rooms a year ahead (each room on each day is an offer, making thousands of offers); and finally an online store easily offers upwards of a million products.

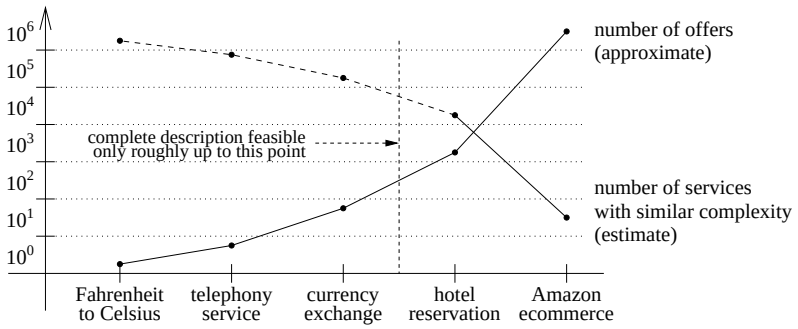

Fig. 3. Numbers of offers of various types of Web services

[3] Found through Web service registries `http://seekda.com` and `http://xmethods.net`

[4] The figure is only illustrative, it is not based on any concrete quantitative analysis.

The figure also shows an estimated potential number of distinct services with the indicated complexity (in terms of the number of offers). We can expect tens of online stores with millions of products, but already for the complexity of thousands of offers we can expect many more services — hotels and travel reservation services being prime examples. The large projected numbers of services with lower complexity can be justified by the vast variety and complexity of the Web and its diverse domains of discourse; nevertheless our uncertainty is higher on the left side of the curve of the potential number of services.

For simpler services, it is not a problem to publish all the relevant information in the semantic descriptions. Complete description becomes unwieldy for services on the right side of the graph: some currency exchange services do not bother to publish the list of supported currencies, hotels only publish up-to-date room availability to their close partners, and Amazon does not provide a "browse-all" functionality at all since it would be highly impractical. The dashed vertical line is a rough threshold above which complete description becomes infeasible.

The reasons against complete semantic description of all offers can be categorized as follows:

Processing performance: for a larger online store, the full product catalogue would make a Web service description impractical or impossible to process, considering current reasoning performance.

Description updates: updating the description in a service registry upon every inventory, availability or pricing change would lead to heavy resource utilization in the registries.

Trade secrets: a full description of service offers could even reveal sensitive stategy information or trade secrets.

While reasoning performance may improve, and registry updates can be optimized, sensitive information and trade secrets will not go away. For instance, a bank service description would have to detail all loan approval procedures in its complete offer description. For banks, the loan approval process with all its considerations is part of what makes some banks successful and others bankrupt. And sensitive information is not limited to such clear cases as banks. Even online retailers such as Amazon do not want to publish all offers, including bundle discounts (e.g., get Harry Potter 7 cheaper with other books in the series). Publishing all the prices and discounted offers in a single, easily accessible place, would provide the competition with insights into Amazon's strategy, and lower Amazon's competitive advantage.

In our experience, the complexity of complete service descriptions is a practical barrier to adoption of SWS technologies within the Web services industry. The need to maintain the complete descriptions, and to include possibly sensitive data, raises the barrier even higher. In other words, service discovery based solely on complete static semantic descriptions is of limited usefulness. On the other hand, less detailed "semantic-light" descriptions (for instance, Amazon would be described as selling books, movie DVDs, music CDs etc.) limit the SWS automation to simple but imprecise matchmaking and ranking.

These limitations of static semantic Web service discovery affect adversely the adoption of semantic technologies, and the lack of automation without semantics in turn

lowers the adoption of Web service technologies themselves. While it may seem from Figure 3 that only a relative minority of Web services cannot be described completely, these complex services represent a significant economical value among the (potentially) available Web services.

3 Semantic Web Service offer discovery

As we have shown, static service discovery based on complete descriptions is, in many important cases, not feasible. Therefore, we split the task of finding the most appropriate offer from all the available Web services into static *Web service discovery* followed by dynamic *offer discovery*. The static Web service discovery uses coarse-grained semantic Web service descriptions to find services that potentially match the user's goal, and the dynamic offer discovery uses the semantic description of the Web service interface to automatically find any appropriate offers. With offer discovery, the set of steps can be rephrased as follows:

1. Web service discovery — find all the available Web services that may be able to fulfill the user's goal (i.e. discard those which, based on their description, cannot fulfill the user's goal).
2. Offer discovery — by interacting with the discovered services, find all their offers relevant for the goal.
3. Filtering — filter out offers that do not fit the user's constraints.
4. Ranking, selection — rank the remaining offers based on the user's preferences, and select one to be invoked.
5. Invocation — use the selected service.

Offer discovery can be seen as information retrieval (search) or as negotiation, as discussed in Section 4. Semantic offer discovery should be able to communicate with any Web service and find information about offers relevant to the user's goal. For communicating with the Web services, the offer discovery engine needs a description of the service interface (what operations it contains that can be used to gather offer information) and a description of the exchanged data, to understand the offers and be able to compare it with the goal. In other words, offer discovery needs different semantic description than Web service discovery; the latter needs to know what the service offers, whereas offer discovery needs to know how to talk to the service to get the information.

Seen as a black box, an offer discovery engine has as its inputs the user goal and the set of discovered Web services, and the output is the set of offers which should be of the same granularity as the user goal, even though the semantic description of the service is on a higher level of abstraction (more coarse-grained). For instance, the semantic description for the Amazon e-commerce service could say "this service sells books". For a concrete user goal "buy the last Harry Potter book", the offers could be "Harry Potter 7, Hardback, $12.99" and "Harry Potter 7, Paperback, $8.99".

Offer discovery complements Web service discovery in situations where the latter alone is not feasible. Our main hypothesis is that *the semantic description necessary for automated offer discovery is significantly easier to create and manage (and more acceptable) than the complete semantic description of all the offers*. A further hypothesis

is that *the process of offer discovery is more efficient or on par with the processes of managing the complete descripton and reasoning with it in Web service discovery*. The terms "easy", "acceptable" and "efficient" are defined more clearly in Section 6, where we propose evaluation criteria for offer discovery solutions.

We should note that what we call *offer discovery* is elsewhere in literature (e.g. [6]) 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".

4 Related work

Semantic Web service offer discovery, as defined in the preceding section, is related to earlier research in automated negotiation, and to the related areas of query processing and information gathering.

The term *negotiation* has been used for different purposes in a variety of computer science fields, e.g. electronic commerce, grid computing, distributed artificial intelligence and multi-agent systems. In electronic 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 query answering and information retrieval has dealt, among others, with using multiple information sources to gather the requested (or relevant) information (cf. [7]), 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. The particular problem in SWS offer discovery is the description of the services and their operations so that information retrieval techniques would be applicable.

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 any 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.

Apart from related work described above, we know of only one published attempt that involves dynamic offer discovery in Semantic Web Services: Zaremba *et al.* [10, 11] talk about a so-called "contracting interface" with a described choreography. In

their case, the SEE client follows the predefined choreography to find out the concrete price offered by a discovered Web service. The contracting interface can be likened to a prescribed protocol for offer discovery.

5 Envisioned solution components

While we do not have a working solution at this time, we can describe the major components necessary for any solution for semantic Web service offer discovery. Offer discovery takes a number of discovered Web services (or their descriptions, to be more precise) together with a user goal and returns a set of offers from these Web services. Any of the discovered Web services can provide any number of relevant offers, or none at all, and at least initially we can assume that offers from different services are independent. Therefore, we describe offer discovery in terms of dealing with a single service; if multiple services have come from Web service discovery, we can deal with each one of them separately. Offer discovery can be seen as a three-step process:

1. selecting the "offer-inquiry" operations from among all the operations of the discovered Web service;
2. planning the execution of some or all of these operations, based on what data is available in the user's goal and what data the operations return;
3. invocation of the selected operations according to the plan, translating the appropriate goal data into the appropriate XML messages.

The first step is necessary because we cannot assume that all the operations of any Web service can be used in offer discovery. Indeed, Web service interfaces often intermix operations for offer inquiry with operations that actually provide the resulting product or service, for instance a hotel service would mix the availability inquiry operations with the operations for booking rooms. For purpose of automatic invocation, we must select operations that are *safe* in the same sense in which the Web architecture [1] defines "safe interactions":

> A *safe interaction* is one where the agent does not incur any obligation beyond the interaction. An agent may incur an obligation through other means (such as by signing a contract). If an agent does not have an obligation before a safe interaction, it does not have that obligation afterwards.[5]

To recognize the "offer-inquiry" operations, we need semantic annotations about the nature of operations. In step with the Web architecture, WSDL 2.0 has a mechanism for annotating Web service operations as safe; it is unclear whether more information would be necessary for selecting "offer-inquiry" operations; if so, they can be added

[5] A canonical example of a safe interaction is information retrieval — the client may query a service about the availability of a hotel room, yet by issuing the query the client makes no commitment to book the room. Note that safety is not the same as idempotency: safe operations are generally idempotent, but idempotent operations need not be safe — for example a delete operation on a data store is idempotent but not safe.

using SAWSDL[6] (Semantic Annotations for WSDL and XML Schema), a specification from the W3C; specifically using `modelReference` on WSDL operations. It remains to be seen whether all safe operations can be seen 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.

When we have selected the suitable operations, we can use information gathering and planning techniques in the second step to plan an execution that will return relevant information about the offers pertinent to the user's goal. We do not have a firm definition of *relevant* at the moment, and we expect that this process can even involve some heuristics for optimizing the interactions with the Web service. Retrieving too little information will give us an incomplete view of the offers, whereas retrieving too much information would be overhead and a potential performance problem. This step requires semantic annotations of the operation inputs and outputs; in other words, what parameters the operations require and what kind of information they return. Such annotations can be added as SAWSDL `modelReference`s on the XML schema elements that are the input and output messages of the Web service operations.

The third step actually executes the plan and invokes the operations. Its role is to ground the goal data (presumably in a Semantic Web language, e.g. RDF or in WSML [4]) to the XML messages expected by the operations, and interpret the returned XML messages as semantic data about offers. This step can probably be implemented with simple reuse of some automatic Semantic Web service invocation mechanism that implements the grounding (cf. [8]), and it needs annotations that specify the data grounding transformations.

6 Evaluating SWS offer discovery approaches

The expected end contribution of our work is an efficient approach to automatic offer discovery that complements Web service discovery based on static descriptions. Even though we do not yet have a concrete solution, we can sketch the ways in which we expect to evaluate it. The evaluation criteria listed below are independent of the details of any proposed solution.

The efficiency of an offer discovery approach is evaluated in two dimensions: volume and complexity of the necessary semantic description, and the performance and scalability of the discovery process, as described below. The offer discovery engine will form a part of a semantic execution environment (SEE). The use cases for a SEE include the common scenarios of electronic shopping and travel scheduling[7], but also existing real-world applications such as e-Government Emergency Planning, as described in [3], and any such use cases should be helpful in testing offer discovery.

The major benefit of the presence of offer discovery in a SEE is that the semantic descriptions of Web services need not be complete and detailed (e.g. the whole catalogue of an online store). This guides the first evaluation criterion: *the semantic description necessary to enable automated offer discovery must be significantly simpler than a complete and detailed static semantic description of the service.* The relative simplicity of

[6] http://w3.org/TR/sawsdl

[7] See http://sws-challenge.org/

the semantic descriptions can be tested using reasoning performance comparisons, and user evaluations of the process of creating and managing the full static description vs. the descriptions necessary for automatic offer discovery. As these evaluations require experts, instead of a simple survey with many participants who may not be so expert, we suggest to use the Delphi method [9] which is shown to have good results with fewer participants, even though the process is more time consuming.

Apart from the complexity of the semantic descriptions, offer discovery requires interaction with the Web services, whereas static service discovery based on complete semantic-heavy descriptions requires that the service provider updates the description on every change. Therefore, the second evaluation criterion is: *the reasoning and networked interaction during offer discovery should have better performance and scalability than the combination of reasoning with complete descriptions and network interactions for description updates*. Performance can be compared on specific test cases, and scalability needs to evaluate how the approaches can deal with many services and many service offers. Further, the comparison combines reasoning tasks with network interactions; therefore it is crucial to evaluate different settings of reasoning power vs. networking setup.

In short, the evaluation of SWS offer discovery approaches is in comparison to static service discovery with complete descriptions (complete enough to get comparable results), and it involves experiments in controlled environments for comparing the performance, and expert surveys for comparing the relative simplicity and maintainability of the involved semantic descriptions.

7 Conclusions

Since Web service discovery cannot always be based on complete and detailed semantic description, it needs to be complemented with automatic offer discovery. In this paper, we have described the problem, sketched the components of a solution and the evaluation methodology. Eventually, we intend to develop an approach to SWS offer discovery that will significantly simplify the needed semantic descriptions and thus help ease the adoption of SWS technologies in the industry.

Any offer discovery approach needs to answer the following major questions: how should the user goal and concrete offers be modeled semantically to enable a generic algorithm for offer discovery; and how to select Web service operations that can be used for retrieving information about concrete offers pertaining to the user goal, plus how to sequence the invocations of these operations. There are two concrete steps ahead of us now: we intend first to work on a prototype whose function will help us understand and formalize offers and goals; when these terms are formalized, we can proceed to specify and evaluate a concrete fully-fledged approach to offer discovery.

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. R. Davies. Summative report on potential applications of SWS in eGovernment, 2006. Deliverable D9.16, project DIP (FP6 - 507483), available at `http://dip.semanticweb.org/documents/D916SummativeRpt_potentialAppssSWS_EGov_final.pdf`.
4. J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The Web Service Modeling Language WSML: An Overview. In *Proceedings of the 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science, LNCS*. Springer, 6 2006.
5. 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.
6. 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.
7. 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.
8. J. Kopecký, D. Roman, M. Moran, and D. Fensel. Semantic Web Services Grounding. In *Proc. of the Int'l Conference on Internet and Web Applications and Services (ICIW'06), Guadeloupe*, February 2006.
9. H. A. Linstone and M. Turoff, editors. *Delphi Method: Techniques and Applications*. Addison-Wesley, 1975.
10. 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.
11. M. Zaremba, T. Vitvar, M. Moran, and T. Hasselwanter. WSMX Discovery for SWS Challenge. SWS Challenge Workshop, Athens, Georgia, USA, November 2006.