Supporting Dynamics in Service Descriptions - The Key to Automatic Service Usage

Ulrich Küster and Birgitta König-Ries

Institute of Computer Science, Friedrich-Schiller-Universität Jena
D-07743 Jena, Germany
ukuester,koenig@informatik.uni-jena.de

Abstract. In realistic settings, service descriptions will never be precise reflections of the services really offered. An online seller of notebooks, for instance, will most certainly not describe each and every notebook offered in his service description. This imprecision causes poor quality in discovery results. A matcher will be able to find potentially matching services but can give no guarantees that the concrete service needed will really be provided. To alleviate this problem, a contract agreement phase between service provider and requester following the discovery has been suggested in the literature. In this paper, we present an approach to the automation of this contracting. At the heart of our solution is the possibility to extend service descriptions with dynamically changing information and to provide several means tailored to the abilities of the service provider to obtain this information at discovery time.

1 Introduction

In recent years two trends – the Semantic Web and Web Services – have been combined to form Semantic Web Services, services that are semantically annotated in a way that allows for fully or semi automated discovery, matchmaking and binding. At the core of Semantic Web Services is an appropriate service description language that is on the one hand expressive enough to allow for precise automated matchmaking but on the other hand restricted enough to support efficient processing and reasoning. Several frameworks have been proposed (among them WSMO[1] and OWL-S[2]) but overall there is no consensus about the most suitable formalism yet.

One of the challenges in the design of semantic service description languages and matchmaking tools is the granularity at which services are to be described and thus the precision that can be achieved during discovery. Most efforts describe services at the interface level, e.g. a service that sells computer parts will be described exactly like that. This is fine to service requests that are given at the same level of detail, like *"I'm looking for a service that sells notebooks"*. Such a description is sufficient, if the aim of discovery is to find *potentially* matching services. In this case, the user or her agent will be presented with a list of notebook sellers and will then browse through their inventory or use some other mechanisms to obtain more precise information about available notebooks, their

B. Krämer, K.-J. Lin, and P. Narasimhan (Eds.): ICSOC 2007, LNCS 4749, pp. 220–232, 2007.

configurations and prices. The user will then have to make a decision which notebook to buy and call the service to actually do so. This works fine, as long as the user is in the loop during the process. It does *not* work anymore if complete automation is expected. Take for instance a fine grained and precise request like *"I want to buy a 13 - 14 inch Apple MacBook with at least 1 GB RAM and at least 2.0 GHZ Intel Core 2 Duo Processor for at most $1500"*. Such a request might be posed by a user, but might as well be posed, e.g., by an application run by the purchasing department of a big company with the expectation that matching services will be found, the best one selected and then invoked autonomously by the discovery engine. To handle such requests successfully, detailed information about available products, their properties and their price is needed in the offer descriptions.

In [3] Preist writes about this issue. He distinguishes *abstract, agreed and concrete services*. A concrete service is defined as "an actual or possible performance of a set of tasks [...]" whereas an abstract service is some set of concrete services and described by a machine-understandable description. Ideally, these descriptions are *correct* and *complete*. *Correct* means that the description covers only elements that the service actually provides. The description "This is a notebook selling service" is *not* correct, since the service will typically not be able to deliver all existing notebooks. *Complete* means, that everything the service offers is covered by the description. While completeness of descriptions is rather easy to achieve, correctness is not. To achieve this characteristic much more information would have to be included in the static descriptions published to the service repositories than is usually feasible for several reasons. Typical services will sell hundreds of products and big players may even sell thousands of thousands of different articles. Regardless of whether one creates few comprehensive descriptions each including many products or many specific descriptions each comprising only few closely related articles: The overall amount of information that needs to be sent to the registry and that needs to be maintained and updated will be forbiddingly big. Scalability issues will become increasingly bad when properties of articles like availability, predicted shipping time or prices change dynamically, which will often be the case. Furthermore, many service providers will not be willing to disclose too much information to a public repository. First, a huge up-to-date database of information about products constitutes a direct marketing value that providers will not be willing to share. Second, information about the availability of items may give bargaining power to potential buyers (think of the number of available seats on a certain flight). In these cases – again – the provider will be unwilling to reveal such information.

Preist suggests a separate contract agreement phase following the discovery to determine whether a matching abstract service is really able to fulfill a given request and to negotiate the concrete service to provide. We argue, that similar to discovery, contracting can be automated, if enough information is made available to the matchmaker. To enable such an extension of a matchmaker, a flexible way to represent dynamically changing information in service descriptions and

to obtain this information during discovery is needed. Existing frameworks for semantic web services offer little if any support for this step.

In this paper we extend our previous work on semantic service description languages and semantic service matchmaking in this direction. We propose means to enable fully automated yet flexible and dynamic interactions between a matchmaker and service providers that are used during the matchmaking process to gather dynamic information that is missing in the static service description (like the availability and the current price of certain articles). We then illustrate how this is used to enable contracting within the discovery phase.

The rest of the paper is organized as follows: In Section 2 we provide some background information about the service description language and the matchmaking algorithm used to implement our ideas. Building on that we present the above mentioned extensions to the language and matchmaking algorithm in Section 3. There, we also illustrate and motivate our extensions via a set of examples. In Section 4 we give more details on the implementation of our ideas and how we evaluated it. Finally, we provide an overview of the related work in Section 5 and summarize and conclude in Section 6.

2 DIANE Service Description

In this section we provide some background information about the DIANE Service Description (DSD) that we used to implement our ideas and the DIANE middleware built around it to facilitate the efficient usage of the language.

DSD is a service description language based on its own light-weight ontology language that is specialized for the characteristics of services and can be processed efficiently at the same time. In the following an intuitive understanding of DSD and the DSD matchmaking algorithm is sufficient for this paper. We will therefore include only the necessary aspects of DSD to make this paper self-contained. The interested reader is referred to [4,5,6] for further details. Figure 1 shows relevant excerpts of a DSD request to buy a notebook roughly corresponding to the one mentioned in the introduction in an intuitive graphical notation.

The basis for DSD is standard object orientation which is extended by additional elements, two of which are of particular importance in the context of this paper.

Aggregating Elements: A service is typically able to offer not one specific effect, but a set of similar effects. An internet shop for instance will be able to offer a variety of different products and will ship each order to at least all national addresses. That means, services offer to provide one out of a set of similar effects. In Preist's terminology, these are *abstract* services. Requesters on the other hand are typically looking for one out of a number of acceptable services, but will have preferences for some of these. (Our notebook buyer might prefer a 13-inch, 1.5 GB RAM notebook over a 14-inch, 1 GB RAM notebook – or the other way round.) Thus, DSD is based on the notion of *sets*. Sets are depicted in DSD by a small diagonal line in the upper left corner of a concept. This way, offers

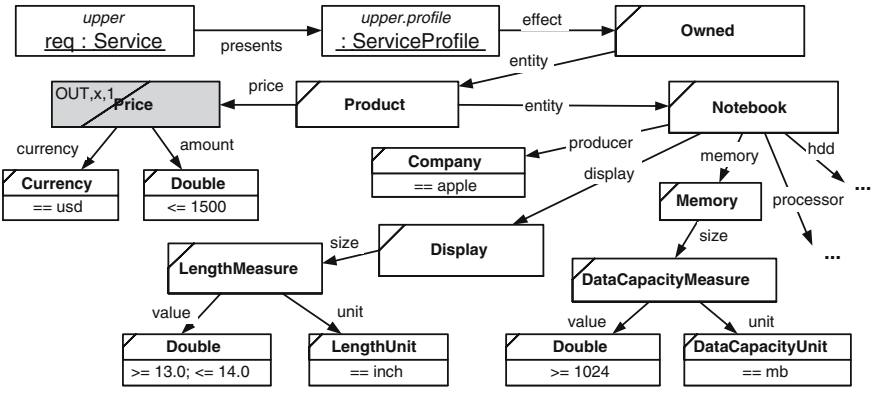

Fig. 1. Simplified DSD request

describe the set of possible effects they can provide and requests describe the set of effects they are willing to accept together with the preference among the elements of this set. Like all DSD concepts sets are declaratively defined which leads to descriptions as trees as seen in Figure 1. This request asks for any one service that sells a notebook that is produced by Apple, has a display size between 13.0 and 14.0 inches, at least one gigabyte RAM and costs at most $1500. Fuzzy instead of crisp sets may be used in requests to encode preferences, e.g., for more RAM or lower cost or to trade between different attributes, e.g. between cost and memory size [6].

Variables: While a service will offer different effects, upon invocation, the requester needs to choose or at least limit which of these effects the service should provide in this specific instance. This step is referred to by Preist as contract agreement and negotiation. Meanwhile, after service execution the requester might need to receive results about the specific effect performed by the invocation. In DSD, *variables* (denoted by a grayed rectangle) are used to support this. In offers, variables are used to encode input and output concepts, in requests (as in in Figure 1) they can be used to require the delivery of certain outputs of the service invocation (like the exact price of the purchased notebook in the example) or to enable configurable request templates.

In our envisioned setting we use a request-driven paradigm for semantic service discovery and matchmaking and assume the need for complete automation. Service providers publish their offer descriptions to some kind of repository. Service users create request descriptions that describe desired effects. The task to find, bind and execute the most suitable service offer for a given request is then delegated to a fully automated semantic middleware that works on behalf of the requester.

Since the semantic middleware is supposed to work in a fully automated fashion up to the invocation of the most suitable offer (if there is one), matchmaking is not limited to identifying potential matches. Instead it has to prepare the invocation by configuring the offers (i.e. choosing values for all necessary inputs) in

a way that maximizes their relevance to the request and it has to guarantee that any identified match is indeed relevant. Note, that this means that we have to carry out contracting in an automated fashion. Because of complete automation the matchmaker has to act conservatively and dismiss the offer in case they are underspecified (for instance if a computer shop doesn't precisely state whether it can provide a particular notebook). The extensions proposed in this paper ensure, that this will happen as seldom as possible.

We use a request-driven approach to matchmaking. Our matchmaker traverses the request description and compares each concept with the corresponding concept from the offer at hand. The degree of match (matchvalue) of two particular concepts is determined by applying any direct conditions from the request to the offer concept and combining the comparison of their types with the matchvalues of the properties of the concepts. These are determined recursively in the same fashion. When the matchmaker encounters a variable in the offer during its traversal of the descriptions it determines the optimal value for this variable with respect to the request. Due to space limitations please refer to [4,6] for further detail.

3 Dynamic Information Gathering for Improved Matchmaking

In this section we will present how we integrated an automated contracting phase into our matchmaking algorithm. We will illustrate our approach by means of offer descriptions for three fictitious providers that are potentially relevant for a user seeking to buy a notebook with particular properties. Note that for reasons of simplicity and due to space limitations all examples shown in this sections have been simplified and show relevant excerpts of the offer descriptions only. In particular all aspects related to actually execute any service operations (grounding to SOAP, lifting and lowering between DSD and XML data, etc.) have been omitted[1].

In order to interact with services, DSD supports a two-phase choreography: An arbitrary number of so-called *estimation steps* is followed by a single *execution step*. The execution step is the final stateful service invocation that is used to create the effect that is desired by the requester. It will be executed for the most suitable offer after the matchmaking is completed. Contrary, the estimation steps will be executed on demand for various offers during the matchmaking process to gather additional information, i.e. to dynamically obtain a more specific and detailed offer description where necessary. Since they may be executed during the matchmaking process for various providers they are expected to be *safe* as defined in Section 3.4. of the Web Architecture [8].

Concepts in an offer description may be tagged as `estimate n out` variables, thereby specifying that they can be concretized and detailed by invoking the

[1] Information about the automated invocation of services described using DSD can be found in [7].

operation(s) associated with the nth estimation step providing the values of the concepts tagged as `estimate n in` variables as input. In order to be able to invoke the corresponding operation, it has to be assured that those variables have already been filled by the matchmaker. Thus our matchmaking algorithm uses a two phase approach[2]:

The first phase is used to filter irrelevant offers, fill variables in the remaining offer descriptions and in particular to collect information about whether a certain estimation step needs to be performed at all. We are able to do this by exploiting the fact that our structured approach to matchmaking (see Section 2) allows us to precisely determine those parts (or aspects) of two service descriptions that did not match. If the matcher encounters a concept that is tagged as `estimate out` variable three cases can be distinguished:

- A perfect match can be guaranteed using the static description alone: A provider states that the shipping time for all offered products does not exceed one week and that the precise shipping time of a particular product can be obtained dynamically. If the request does not require the shipping time to be faster than one week, there is no need to inquire the additional information.
- A total fail is unavoidable from the static description alone: A provider states that the notebooks offered have a price range from $1500 to $2500 and that the precise price can be inquired dynamically. If the requester is seeking a notebook for less than $1500 this offer will not match in any case and may be discarded.
- In all other cases the estimation operation needs to be performed. Even in cases where an imperfect match can be guaranteed based on the static description alone, more precise information obtained dynamically might improve the matchvalue of the service.

After the first phase the necessary estimation operations for all remaining offers will be performed. It is important to stress again that our matchmaking algorithm allows us to determine whether a particular estimation operation offers useful information for a given matchmaking operation. Thus we are able to reduce the expensive execution of estimation operation to the absolute minimum. Once the estimation operation have been executed the corresponding service descriptions will be updated with the newly obtained information. Based on the completed offer descriptions the matchmaker computes the precise matching value of each offer in a second matching phase. This procedure is used in the following example.

Midge, a small internet-based merchant, is specialized on delivering highly customized notebooks. Customers may choose the various components (display, CPU, RAM, HDD, ...) to select a machine that corresponds most closely to their specific requirements. The available components are precisely described in the static offer description but the price depends on the chosen configuration and can be obtained by calling a specific endpoint with the configuration's key data as input.

[2] Actually it uses three phases due to issues related to automated composition but this is not relevant for this paper. Please refer to [6] for further details.

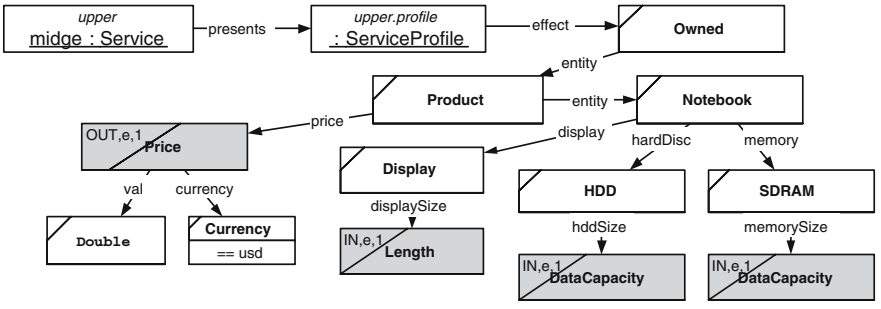

Fig. 2. Relevant parts of Midge's offer description

The relevant part of the offer description of Midge is shown in Figure 2. Midge simply declares the concepts corresponding to the sizes of display, RAM, HDD, ... as `estimate in` variables and the price concept as `estimate out` concept. The matcher will assure to provide single concrete values for the necessary input variables. This way Midge's grounding can be kept extremely simple thereby minimizing the effort involved for Midge to create the service implementation. Note however that the matchmaker will first choose values for display, RAM, HDD, ... and then inquire the corresponding price. Since a general matchmaker does not know typical dependencies between those properties (more RAM usually results in a higher price) it cannot always find the best value for the money. Thus Midge's attempt is lightweight, but not suitable for all cases.

In order to provide greater flexibility and cope with the different requirements of different providers, in this paper, we additionally propose an extended concept of variables to be used in the context of estimation steps. DSD variables – as introduced in [5] – link the inputs and outputs of service operations with concepts in the service descriptions.

For different tasks we suggest four increasingly complex types of binding (or filling) of variables.

- A variable that supports only *simple binding* needs to be filled with a concrete instance value. Examples include the instance *Jena* for a variable of type $Town$ or $< P1Y2MT2H >$ for a variable of type *XSD_Duration*.
- A variable that supports *enumerated binding* can be filled with a a *list* of concrete instances.
- A variable that supports *declarative binding* can be filled with a crisp, declarative DSD set like the set of all towns that are at most 300 km away from Jena or the set of notebooks with a 13 inch display, an Intel Duo Core processor and more than one GB RAM.
- A variable that supports *fuzzy declarative binding* can be filled with a *fuzzy* declarative DSD set. This way preferences can be expressed in a variable filling.

Which type of binding is used in a given service description may depend on a number of factors, including the kind of service, the number of instances

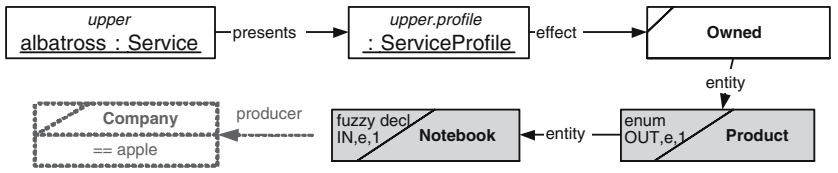

Fig. 3. Relevant parts of Albatross's offer description

associated with certain variables, the willingness of the service provider to share information and its ability to process the more complex bindings. Midge for instance used simple variables, thereby minimizing its effort to create its service implementation but failing to deliver the most suitable offer in all cases. This approach is unsuitable for our next example that will be based on fuzzy declarative and enumerated bindings.

Albatross operates a huge online shop for all kind of electronic products. Including the offers of third party sellers that Albatross's online shop integrates as a broker, hundreds of thousands of articles are available. Human customers can browse the catalogue data through Albatross's website and for the envisioned automated usage Albatross plans to list the most suitable products for a request in a similar fashion. However, in order to limit network bandwidth consumption and to avoid to reveal all the catalogue data, Albatross decided to never list more than thirty items in reply of a single request. It is therefore important for Albatross to carefully select those thirty listings in a way that maximizes the relevance to the request at hand. Since the overall range of available articles is fairly stable, Albatross decided to create a single offer description for each type of article (printer, monitor, notebooks, servers, ...). Depending on the range of articles and the granularity chosen for the descriptions, a fair deal of static information can be included in the descriptions and Albatross's endpoint will not be called for obviously unsuitable requests. To retrieve a maximum of information about the interests of the user, Albatross uses fuzzy declarative binding for the `in` variables of the necessary estimation operation and will retrieve all information available at all about the users preferences. It is up to the implementation of Albatross's endpoint how much of that information is then really evaluated to select the thirty most relevant products (see Section 4 for further detail). The list of these products will be returned in the `out` variable of the corresponding estimation operation, that is therefore declared as an enumerated variable.

The relevant parts of Albatross's offer description for notebooks can be seen in Figure 3. Note that the displayed parts are still very generic. To avoid to be called too many times Albatross could just as well decide to further specialize its descriptions by statically adding further restrictions (e.g. restrict the offer to notebooks produced by Apple as indicated in grey in Figure 3).

One disadvantage of Albatross' descriptions is that due to the broad nature of Albatross' catalogue they can not reveal a whole lot of information statically. This motivated another extension that will be demonstrated with the following example.

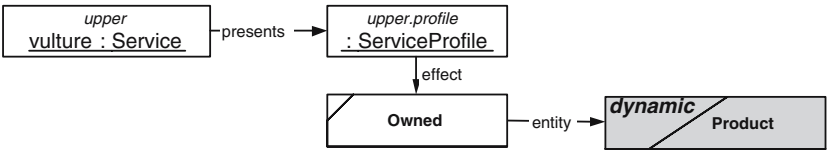

Fig. 4. Relevant parts of Vulture's offer description

Vulture is a reseller of remaining stock and similar extraordinary items. Available items change from day to day and range from hightech servers to wooden lawn seats. Vulture's main interest is to limit maintenance (i.e. programming) cost and to quickly sell any items currently on stock.

The relevant part of the offer description of Vulture is shown in Figure 4. Due to the dynamically changing range of available articles, Vulture cannot provide much information in the static offer description (basically all it states is that Vulture is a vending service). Since such an offer is not very meaningful we created a special operation: Concepts tagged as *dynamic sets* may have an associated estimation operation that will be evaluated right at the beginning of the first phase of the matching process. Since it cannot be assured that input values have been determined by the matchmaker already, the corresponding operation must not have any specific `in` variables. Instead the corresponding concept description from the request will be given as input. In the case of Vulture, Vulture's implementation (see Section 4) simply extracts the type of Product seeked by the requester and returns a listing of all related available products that will be used to complete the offer description during matchmaking. This operation is basically syntactic sugar, the main difference to using standard estimation operations is the time the estimation operation will be evaluated. We chose to add this option since it can pay off to minimize the number of remaining offers early in the matching process (e.g. in cases where automated composition takes place [6]).

4 Implementation and Evaluation

The concepts illustrated above have been implemented in our matchmaker and the supporting DIANE middleware [6]. For the sake of much greater objectiveness we sought to have our ideas and techniques evaluated by a greater community in addition to our own evaluation. We believe the ongoing Semantic Web Services Challenge[3] [9] to be an ideal setting for this task. The challenge presents sets of increasingly difficult problem scenarios[4] and evaluates solutions to these scenarios at a series of workshops. It specifically aims at developing a certification process for semantic service technology.

A basic example similar to the case of Midge has been used in the solutions for the first Shipment Discovery Scenario of the SWS-Challenge to

[3] http://sws-challenge.org
[4] http://sws-challenge.org/wiki/index.php/Scenarios

dynamically obtain the price of a shipping operation depending on the properties of the parcel[10,11]. Our solution including dynamic gathering of the price was successfully evaluated at the second SWS-Challenge workshop in Budva, Montenegro[5]. Since then we improved the integration of the estimation step handling into the matchmaking algorithm in order to avoid the execution of unnecessary estimation operations whereas in our original solution for the SWS-Challenge scenario all available estimation operations were executed. This was motivated by the fact that the inevitable communication cost for the estimation operations quickly dominates the cost of the whole matching process.

We submitted a new scenario[6] to the SWS-Challenge that meanwhile has been accepted as an official scenario by the SWS-Challenge Programm Committee. This scenario deals among other things with the dynamic retrieval of a listing of available products from given web service endpoints to perform service discovery for very specific purchasing requests. We used offer descriptions similar to that of Vulture to model the services and successfully solved the relevant goals A and B of the scenario. At the fourth SWS-Challenge workshop in Innsbruck, Austria our current solution [12] to both scenarios was evaluated as the most complete discovery solution[7]. It can be downloaded from the SWS Challenge web site[8].

To test our solution under more realistic settings in terms of the number of available products we simulated the Albatross service. We gathered about 2500 structured descriptions of notebook offerings with a total of more than 100.000 attributes from the Internet and stored it in a relational product database similar to one that a service like Albatross might have. We created a corresponding offer description and a service implementation that operates on our database. We applied the restriction that no more than thirty offers will be listed for any request and created a grounding of our service description to our implementation that queries our product database and uses the fuzzy information taken from the request to select the thirty notebooks expected to be most relevant. We ran a series of requests against the Albatross offer and measured the time needed to gather the relevant notebooks. We ran the experiment locally and did not measure communication cost. On an Intel Pentium 1.8 GHz machine it took about 500 ms to query the product database and convert *all* 2500 notebook descriptions including their attributes into ontological DSD instances. On average it then took another 950 ms to determine the precise matchvalue of all these notebooks with respect to the fuzzy description taken from the request. Based on that we returned the 30 top-ranked products and could be sure that these were the most relevant with regard to the request as determined by the matchmaker on behalf of the user. These results show the practical applicability of our approach for the given setting. This is particular true since there is much room for optimization left. One could, e.g., include hard restrictions from the

[5] http://sws-challenge.org/wiki/index.php/Workshop_Budva#Evaluation
[6] http://sws-challenge.org/wiki/index.php/Scenario:_Discovery_II_and_Simple_Composition
[7] http://sws-challenge.org/wiki/index.php/Workshop_Innsbruck#Evaluation
[8] http://sws-challenge.org/wiki/index.php/Solution_Jena

request (like a price limit or a limitation to specific brands) in the query to the database, thereby drastically reducing the number of notebooks that need to be converted to DSD instances and dealt with in the first place.

5 Related Work

To the best of our knowledge, none of the matchmaking algorithms based on ontological reasoning like the ones proposed by Kaufer and Klusch [13], Klusch et al. [14], Li and Horrocks [15] or Sycara et al. [16] support an automated contracting phase with dynamic information gathering as we propose in this paper.

Keller et al. [17,18] propose a conceptual model for automated web service discovery and matchmaking that explicitly distinguishes between abstract service discovery and service contracting. The latter refers to inquiring the dynamic parts of service descriptions and negotiating a concrete contract. Unfortunately, only an abstract model is presented and no details or ideas about an implementation are provided.

Similar to our approach [19] acknowledges the need to support highly configurable web services but focuses on algorithms for the optimal preference-based selection of configurations for such web services and not on how to dynamically explore the value space of possible configurations.

As mentioned in Section 4 the two discovery scenarios of the SWS-Challenge require the dynamic inquiry of price information and product listings. At the first workshop in Budva[9] DIANE was the only submission able to dynamically inquire price information. Meanwhile two teams successfully improved their technology to address this issue[10] [20,21]. Both solutions are similar to ours in that they complement service descriptions with information about operations that can be executed in order to dynamically gather additional information. However, both do not automatically evaluate whether a particular information needs to be gathered and therefore execute *all* available information retrieval operations. They are also lacking an equivalent of our concept of binding types for variables to accommodate for different information needs.

The work that is most closely related to the work in this paper is the Web Services Matchmaking Engine (WSME) proposed by Facciorusso et al. [22]. Like our approach, WSME allows to tag properties of a service description as dynamic, to provide means to dynamically update those properties by calling the service provider's endpoint and to pass properties from the request in that call to allow the provider to tailor the response. However, dynamic service properties are always evaluated whereas in our work we are able to detect whether a particular evaluation is needed or not. Furthermore, service descriptions in WSME are flat (based on simple name-value property pairs) and do not support fine-grained ranking as DSD does, therefore the WSME does not aim at complete

[9] http://sws-challenge.org/wiki/index.php/Workshop_Budva#Evaluation
[10] http://sws-challenge.org/wiki/index.php/Workshop_Innsbruck#Evaluation

automation and leaves the user in the loop. To avoid this was one of the main motivations of our work.

6 Summary and Conclusion

In order to allow for fully automated usage of service oriented architectures, it must be possible to automatically select, bind, and invoke appropriate services for any given request. This automation is hampered by the unavoidable imprecision in offer descriptions. Typically, discovery will only be able to find *possibly* matching services. It needs to be followed by a contracting phase that makes sure that the required service can indeed be rendered. In this paper, we have presented an approach that allows for the integration of the contracting into the discovery phase and for its complete automation. The approach is based on augmenting service descriptions with dynamic parts. Information about this parts can be obtained from the service provider at discovery time.

References

1. de Bruijn, J., Bussler, C., Domingue, J., Fensel, D., Hepp, M., Keller, U., Kifer, M., König-Ries, B., Kopecky, J., Lara, R., Lausen, H., Oren, E., Polleres, A., Roman, D., Scicluna, J., Stollberg, M.: Web service modeling ontology (WSMO) (W3C Member Submission June 3, 2005)
2. Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., Sirin, E., Srinivasan, N., Sycara, K.: OWL-S: Semantic markup for web services (W3C Member Submission November 22, 2004)
3. Preist, C.: A conceptual architecture for semantic web services (extended version). Technical Report HPL-2004-215 (2004)
4. Klein, M., König-Ries, B.: Coupled signature and specification matching for automatic service binding. In: Zhang, L.-J(L.), Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, Springer, Heidelberg (2004)
5. Klein, M., König-Ries, B., Müssig, M.: What is needed for semantic service descriptions - a proposal for suitable language constructs. International Journal on Web and Grid Services (IJWGS) 1(3/4) (2005)
6. Küster, U., König-Ries, B., Klein, M., Stern, M.: Diane - a matchmaking-centered framework for automated service discovery, composition, binding and invocation on the web. International Journal of Electronic Commerce (IJEC), Special Issue on Semantic Matchmaking and Retrieval (2007) (to appear)
7. Küster, U., König-Ries, B.: Dynamic binding for BPEL processes - a lightweight approach to integrate semantics into web services. In: Second International Workshop on Engineering Service-Oriented Applications: Design and Composition (WESOA06) at ICSOC06, Chicago, Illinois, USA (2006)
8. Walsh, N., Jacobs, I.: Architecture of the world wide web, volume one. W3C recommendation, W3C (2004), `www.w3.org/TR/2004/REC-webarch-20041215/`
9. Petrie, C.: It's the programming, stupid. IEEE Internet Computing 10(3) (2006)
10. Küster, U., König-Ries, B., Klein, M.: Discovery and mediation using diane service descriptions. In: Second Workshop of the Semantic Web Service Challenge 2006, Budva, Montenegro (2006)

11. Küster, U., König-Ries, B.: Discovery and mediation using diane service descriptions. In: Third Workshop of the Semantic Web Service Challenge 2006, Athens, GA, USA (2006)
12. Küster, U., König-Ries, B.: Service discovery using DIANE service descriptions - a solution to the SWS-Challenge discovery scenarios. In: Fourth Workshop of the Semantic Web Service Challenge - Challenge on Automating Web Services Mediation, Choreography and Discovery, Innsbruck, Austria (2007)
13. Kaufer, F., Klusch, M.: WSMO-MX: a logic programming based hybrid service matchmaker. In: ECOWS2006. Proceedings of the 4th IEEE European Conference on Web Services, Zürich, Switzerland, IEEE Computer Society Press, Los Alamitos (2006)
14. Klusch, M., Fries, B., Khalid, M., Sycara, K.: OWLS-MX: Hybrid OWL-S Service Matchmaking. In: Proceedings of the First International AAAI Fall Symposium on Agents and the Semantic Web, Arlington, Vriginia, USA (2005)
15. Laukkanen, M., Helin, H.: Composing workflows of semantic web services. In: Workshop on Web Services and Agent-based Engineering, Melbourne, Australia (2003)
16. Sycara, K.P., Widoff, S., Klusch, M., Lu, J.: Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. Autonomous Agents and Multi-Agent Systems 5(2) (2002)
17. Keller, U., Lara, R., Lausen, H., Polleres, A., Fensel, D.: Automatic location of services. In: Gómez-Pérez, A., Euzenat, J. (eds.) ESWC 2005. LNCS, vol. 3532, Springer, Heidelberg (2005)
18. Fensel, D., Keller, U., Lausen, H., Polleres, A., Toma, I.: WWW or what is wrong with web service discovery. In: W3C Workshop on Frameworks for Semantics in Web Services, Innsbruck, Austria (2005)
19. Lamparter, S., Ankolekar, A., Studer, R., Grimm, S.: Preference-based selection of highly configurable web services. In: WWW2007. Proceedings of the 16th International World Wide Web Conference, Banff, Alberta, Canada (2007)
20. Brambilla, M., Celino, I., Ceri, S., Cerizza, D., Valle, E.D., Facca, F., Tziviskou, C.: Improvements and future perspectives on web engineering methods for automating web services mediation, choreography and discovery: SWS-Challenge phase III. In: Third Workshop of the SWS Challenge 2006, Athens, GA, USA (2006)
21. Zaremba, M., Tomas Vitvar, M.M., Hasselwanter, T.: WSMX discovery for SWS Challenge. In: Third Workshop of the Semantic Web Service Challenge 2006, Athens, GA, USA (2006)
22. Facciorusso, C., Field, S., Hauser, R., Hoffner, Y., Humbel, R., Pawlitzek, R., Rjaibi, W., Siminitz, C.: A web services matchmaking engine for web services. In: EC-Web2003. 4th International Conference on E-Commerce and Web Technologies, Prague, Czech Republic (2003)