Service Discovery using DIANE Service Descriptions - A solution to the SWS-Challenge Discovery Scenarios

Ulrich Küster and Birgitta König-Ries

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

Abstract. In this paper we present how DIANE service descriptions are used to solve all but one goal of the SWS-Challenge[1] service discovery scenarios (status from the call for the fourth SWS-Challenge workshop in Innsbruck, June 2007). We build on our papers for the previous SWS-Challenge workshops [1–3] and focus on how the new goals for the original discovery scenario related to temporal reasoning as well as the goals for the completely new second discovery scenario have been addressed. We also introduce the enhancements made to the DIANE framework in order to be able to cope with these goals.

1 Introduction

The Semantic Web Services Challenge [4] has presented a set of problem scenarios to provide a common application to explore the trade-offs among existing approaches that facilitate the automation of mediation, choreography and discovery for services using semantic annotations. The *mediation scenario* covers the mediation problem to make a legacy order management system interoperable with external systems that use a simplified version of the RosettaNet PIP3A4 specifications[2]. This scenario has not been changed since the last workshop in Athens, GA, November 2006 and therefore our solution[3] to this scenario has not changed either and will not be dealt with in this paper. The first discovery scenario deals with the problem to dynamically discover and invoke the most appropriate shipment service provider for a set of given shipment requests. Our original solution to this scenario which is based on the DIANE service description framework is described in [2, 3]. For the upcoming fourth SWS-Challenge workshop in Innsbruck (June 2007) a set of new goals dealing with temporal reasoning about shipping times has been added to the first discovery scenario and a completely new *second discovery scenario* dealing with the discovery of a vendor for a specific product has been added to the challenge.

[1] www.sws-challenge.org
[2] http://www.rosettanet.org/PIP3A4

In this paper we introduce the enhancements made to the DIANE framework that enabled us to solve the new discovery goals and present our solution in detail. To make this paper self-contained, we will include a brief introduction to the DIANE service descriptions (DSD) in Section 2. Section 3 describes how conditions combining different attributes of a service description can be expressed in DSD and how this new feature of DSD has been used to address the new goals for the first discovery scenario. In Section 4 we cover how the various aspects of the new vendor discovery scenario has been addressed. We explain how incomplete static offer description can be automatically updated with product listings which are retrieved dynamically from the corresponding service providers, how user preferences can be included in service requests and how offers are automatically combined to service more complex requests. Finally, in Section 5 we will evaluate our approach, outline the directions of future work and summarize.

2 What is DSD?

The goal of service-oriented computing is the ability to dynamically discover and invoke services at run-time, thus forming networks of loosely-coupled participants. The most important prerequisite is an appropriate semantic service description language – and with *DIANE Service Description* (DSD) [5, 6] we provide such a language together with an efficient matching algorithm.

One main difference between DSD and other semantic service description languages is its own light-weight ontology language that is specialized for the characteristics of services and can be processed efficiently at the same time. The basis for this ontology language is standard object orientation which is extended by four additional elements:

– Services perform world-altering operations (e.g., after invoking a shipment service, a package will be transported and a bill will be issued) which is captured by *operational elements*. We view this is the most central property of a service, thus, in DSD, services are primarily described by their effects – all other aspects (as flow of information, choreography etc.) are seen as secondary, derived properties. An effect is comprehended as the achievement of a new *state*, which in DSD is an instance from a state ontology.
– Services offer/request more than one effect (e.g. a shipment provider offers shipment to a multitude of possible locations and for various types and sizes of packages) which is captured by *aggregational elements*. Thus, the effect of a service is typically a *set of states*. In DSD, these are declaratively defined which leads to descriptions as acyclic directed graphs (see example in the next section).
– Services allow to choose among the offered effects (e.g. as a matter of course all shipment providers allow to input the package being transported and to select where to pick it up and where to ship it) which is captured by *selecting elements*. In DSD, selecting elements are represented as variables that can be integrated into set definitions, thus leading to configurable sets. Therefore,

a service offer in DSD is represented by its effects as configurable sets of states.

– The appropriateness of services and their effects is varying for different requesters (e.g., in the scenario, a more expensive shipment provider will still be accepted, but a less expensive one will be preferred) which is captured by *valuing elements*. In DSD, these elements are represented by using *fuzzy sets* instead of crisp ones in service request descriptions thereby capturing all preferences of the requester – the larger the membership value the higher the preference.

For processing a semantic service description language, an efficient *matchmaking algorithm* is needed. Thus, for a given DSD offer description o and a given DSD request r, a matchmaker has to answer the following question: How well are o's crisp configurable effect sets contained in r's fuzzy effect sets and which configuration of o's effect sets will maximize this value? Our implementation solves this by stepping through the graphs of o and r synchronously in order to calculate the matching value in [0,1] as well as the optimal configuration of the variables. As the preferences are completely included in r, in contrast to existing approaches, our matcher does not need to apply any heuristics and thus is able to operate deterministically.

In order to interact with a service, DSD supports a simple choreography. During matchmaking several stateless *estimation steps* may be performed where operations of the service are called, which provide information (like the price of a package given its weight) but do not imply any contract between the requester and the provider. After the best match is found that service can be invoked by executing a single *execution step* which is supposed to produce the offered effects.

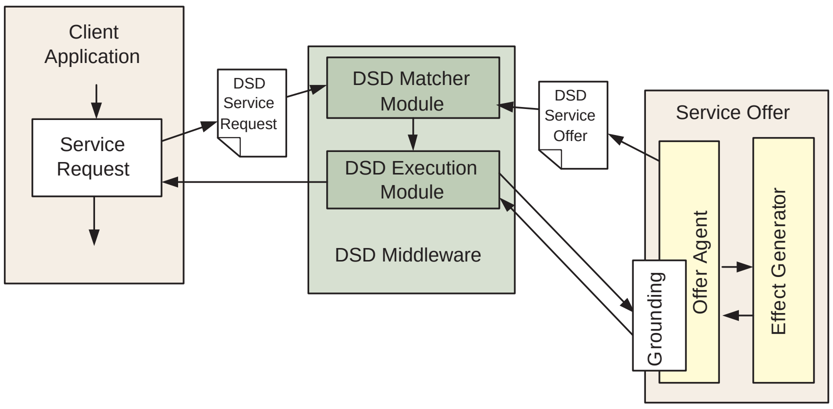

Fig. 1. DSD Middleware

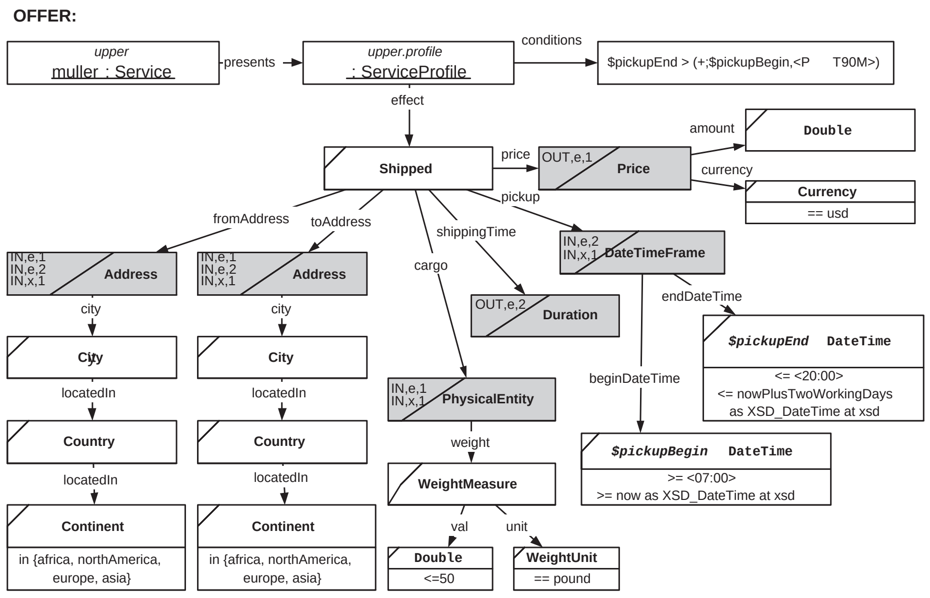

Fig. 2. Excerpt from the DSD description for the Muller shipment service.

The proposed concepts are implemented in the DSD middleware. The overall architecture of the system is depicted in Figure 1. On the left hand side, the client is shown. It runs an application that at some point in time requests an external service to provide some functionality. The service request is formulated using DSD (e.g. by filling a predefined semantic request template) and sent to the middleware. There, the matcher module compares it to the available service offers. When a matching result is found, it is configured appropriately and passed on to the execution module. This module then invokes the offer using the grounding found in the offer description and finally returns the execution results to the client application. More detailed information how to integrate semantic service requests into existing processes using the DSD Middleware can be found in [7].

3 Temporal Reasoning about Shipping Times

The first discovery scenario posed by the SWS Challenge consists of dynamically finding and invoking the most suitable shipment service for a number of shipment requests.

Figure 2 shows the description of Muller, one of the imaginary shippers used in the scenario. The service collects a *cargo* at a certain *pickup* time and ships it from *fromAddress* to *toAddress* for a certain *price*. A detailed discussion of this offer description is beyond the scope of this paper but can be found in [3].

The new temporal reasoning goals for the first discovery scenarios are composed of a set of requests for a shipping operation with an upper limit on the maximum shipping time of the parcel. In addition to the attributes of the other goals (like addresses, cargo, price limits, . . .) these goals state the current time and the number of business days after which the parcel has to be delivered.

The offers allow to input a pickup time frame but pose requirements on the length of this time frame (e.g. at least 90 minutes - Muller), on the overall time during which packages are collected (e.g. between 7 am and 8 pm - Muller) and on the minimum or maximum advance notice for pickup (e.g. at least 1 hour, at most 5 business days - Runner). Additionally they state rules how to compute the shipping time depending on the collection time and whether the shipment is national or international (e.g. Muller: "Ships in 2/3 (domestic/international) business days if collected by 5pm").

Obviously, in order to match one of the temporal resoning goals with one of the available offers the following tasks are necessary: Depending on the current time and the requirements of the shipper, a suitable pickup time frame has to be chosen as input for the offer. Based on the end of that pickup time frame and the address to ship to the expected shipping time of that offer has to be evaluated. Finally this shipping time has to be compared with the corresponding requirements from the request to determine a match or a fail. There were two main difficulties in using DSD for such matching: A lack of direct support for rules and a lack of support to express conditions combining different attributes of a service description. Both issues are covered in the rest of this section.

3.1 Evaluating Rules to Determine the Shipping Time

DSD uses a declarative set based and not a rule based approach to service matching and has no direct support for rules as used by the offers to determine the expected shipping time. In order to overcome this deficiency it is possible to delegate the evaluation of the shipping time to an external service or module by using so-called *estimation operations*. DSD estimation operations can be used to enable dynamic gathering of additional information from a service provider during matchmaking. They were, for instance, used to inquire the price of a shipping operation at the endpoint of the Muller service providing the addresses and the specifications of the cargo as input (see [3]). In the same way estimation operations have been used to delegate the rule-based computation of the shipping time of a particular service to a service endpoint providing the addresses and the pickup time as input. Thus, in Figure 2, the shipping time is declared as output of the second estimation step (`OUT,e,2`) which uses the addresses and the pickup time as input (declared by markers `IN,e,2`). The other similar markers are used to declare the inputs and outputs of the first estimation step (used to inquire the price) and the actual service invocation.

3.2 Using Conditions that Combine Multiple Attributes

The second difficulty lay in expressing the requirements on the pickup time frame length of the shippers. In order to express a condition like *"there must be at least an interval of 90 minutes for collection"* one must express that the difference between the end of the pickup interval and the begin of the pickup interval must be greater than 90 minutes. This requires two features that DSD was lacking at the time of the last SWS-Challenge workshop: arithmetic operations in conditions and conditions combining the values of different attributes (the first does not make sense without the last). In particular the latter is problematic since it breaks the tree structure of DSD descriptions and hinders the local optimization of offer configurations. In order to ensure an optimal offer configuration in the general case one would either have to test all possible configurations of an offer (which is impossible if input values are infinite like in the case of the pickup time) or implement some sort of backtracking in case the chosen configuration of one attribute prevents a successful configuration of another attribute (if the matchmaking algorithm chooses a late start of the pickup time frame it might be impossible to respect the 90 minute interval length requirement when choosing the end of the pickup time frame). To avoid performance issues during the matchmaking our approach is to still choose variable values locally and accept that we might have to dismiss an offer due to an unfortunate choice for some input values.

Overall we made the following extensions to DSD and the DSD matchmaking algorithm to achieve sufficient expressivity to tackle the above mentioned requirements. We extended DSD with the ability to express additional *multi attribute conditions* in service descriptions that reference arbitrary labeled attributes of a description and use arithmetic operations. Muller's requirement was thus expressed as `"$pickupEnd > (+;$pickupBegin, <PT90M>)"` encoding that the end of the pickup time frame must be later than the begin of the pickup time frame plus a 90-minute duration (see Figure 2). During matchmaking, depending which attribute will be matched first, the value for either $pickupEnd or $pickupBegin will be chosen first. At this point the matchmaking algorithm checks whether there are any additional conditions that use this value and will update those conditions with the chosen value. Assume the value `<2007-06-06T14:00:00>` for $pickupEnd had been chosen. In this case only one open reference remains in the condition and the condition will be transformed to `"$pickupBegin < <2007-06-06T12:30:00>"` which will be added as regular condition `"< <2007-06-06T12:30:00>"` to the description of the DSD set labeled $pickupBegin. When the matchmaking algorithm continues to match that attribute it will respect that condition and either choose an appropriate value or dismiss the offer if that is impossible. This approach is efficient but unfortunately not complete: as mentioned above an unfortunate choice for one attribute value might prevent an overall successful configuration of an offer. To improve on this issue without compromising efficiency is clearly a field of future work.

4 Solving the Second Discovery Scenario

The new second discovery scenario deals with finding a vendor for specific products. The scenario description states three distinct sources of difficulty. In this section we will introduce our solution to the scenario by presenting in turn how we addressed each of the mentioned difficulties.

4.1 Dynamic Product Listings

The scenario contains three descriptions of imaginary online shops that sell electronic products. A concrete list of 19 available products is given statically but a listing of available products by product type (like a list of all offered notebooks) can also be obtained by calling a specific operation at the service's endpoints and solutions were supposed to indicate how they would address a more realistic situation with hundreds or thousands of available products. We address this situation by introducing *dynamic offer descriptions*.

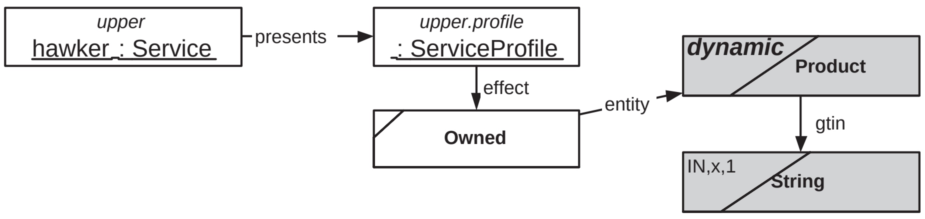

Fig. 3. Excerpt from the DSD description for the Hawker vending service.

Figure 3 shows the relevant excerpts from the offer descriptions of the Hawker vending service (all grounding information has been omitted). Without the listing of the products little information can be included in the offer description. Basically the offer simply states, that Hawker sells products given it's GTIN number as input. 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 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 Hawker, Hawker's grounding simply extracts the type of Product seeked by the requester and then invokes Hawker's endpoint to list the available products of this type. The returned xml listing will be converted to a list of DSD based instances (using mapping information from Hawker's grounding) and Hawker's offer will be annotated with the retrieved instances, thus providing a concrete up-to-date listing of the available products. The matchmaking will then be performed based on that listing. By changing it's grounding and using additional information from the provided request concept beside the type of product requested, Hawker may finetune the procedure in order to avoid to return too long product listings. This way, dynamic product listings could be nicely addressed and the solutions to Goals A1 and A2 were straightforward.

4.2 Competing Request Preferences

Some goals contain different (and sometimes competing) preferences regarding certain attributes of the requested product. Goal C4 for instance requests to buy a notebook stating a price limit and prefers better products as long as that limit is satisfied. Additionally that goal defines a ranking of preferences (e.g. more RAM is more important than a bigger HDD) to detail what constitutes a better product.

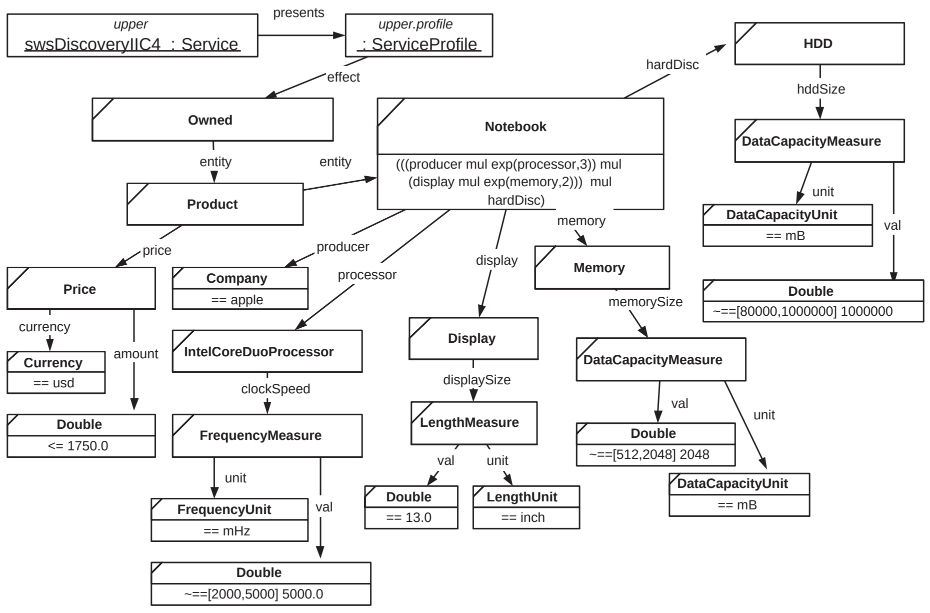

Fig. 4. Excerpt from the DSD description for Goal C4 showing preferences.

DSD is very well suited to capture such finegrained and competing preferences using fuzzy sets. Figure 4 shows that part of Goal C4 that corresponds to the notebook purchasing request. An *fuzzy direct condition* like " `==[2000,5000] 5000`" in the value set of the processor speed attribute is used to build a fuzzy set of Double values. The requested value is 5000, but values from the range $[2000, 5000]$ are included fuzzily with linearly decreasing degree of membership and preference. This encodes that the requester requires the processor speed to be at least 2000 and prefers faster processors. Similar expressions are used for the size of memory and hard drive. To express ranking of preferences (RAM is more important than HDD), *fuzzy connecting strategies* like the one in the notebook set are used. By specifying how to combine the fuzzy membership values of the attributes of a notebook the fuzzy notebook set is constructed. In Figure 4 the values for the attributes are multiplied but the value of the memory attribute is squared and that one of the processor attribute is cubed. This way the influence of those attributes is increased, capturing that processor speed is most important and memory size is more important than hard drive capacity. More information on how to encode user preferences using fuzzy sets can be found in [8, 5].

4.3 Simple Service Composition

The scenario includes four basic types of simple service composition.

Unrelated composition Goal C1 simply asks for multiple products, but these products are not related. This goal can be expressed in DSD by a request with multiple effects, one for each product. The matchmaking algorithm will automatically find a set of offer(-configurations) so that each request effect will be provided exactly once [9].

Correlated composition Goal C2 requests a notebook and a compatible docking station. Thus the two requested products are correlated and cannot be handled seperately. DSD is capable to express such correlations and to correctly compose and configure multiple offers accordingly [9]. Unfortunately, we were nevertheless unable to solve goal C2 correctly. Compatibility of docking stations and notebooks in the scenario is given by a property of each docking station that holds a list of the GTINs[3] of the compatible notebooks. To ensure whether a notebook is compatible to a docking station one has to check whether the notebook's GTIN is contained in the docking stations compatibility list. Currently the DIANE framework lacks sufficient support for matching of list-based attributes to handle this case.

Unrelated composition with global condition Goal C3 requests multiple unrelated products (similar to Goal C1) but states a price limit for the complete purchase. This can be expressed in DSD by using the multi attribute conditions that were introduced in Section 3. This way the goal could be expressed and solved but the same limitations apply. Our algorithm is correct but not complete. Depending on the available products and the concrete request at hand it might happen, that after the first product is chosen, too little money remains to add the other products. In this case the matchmaking will fail and no product will be purchased.

Unrelated composition with global condition and preference Goal C4 simply added preferences on top of the requirements of Goal C3. The encoding of preference has already been covered above and the upper price limit was expressed the same way as for Goal C3.

[3] A globally unique product identifier used in the scenario

5 Evaluation and Summary

In this paper, we have described how DSD and the DIANE middleware have been used to tackle the discovery problems stated by the SWS-Challenge initiative. We focused on how the changes to the problem scenarios made since the last workshop in Athens, GA, November 2006 habe been addressed. These changes included new temporal reasoning goals in the context of the shipment discovery scenario and a new purchasing scenario that requires to dynamically process product listings, to express preferences and to perform some basic composition in order to service the requests.

We introduced the notion of dynamic sets (i.e. offers that are complemented automatically at the beginning of the matching process) and could this way deal very well with the dynamic product listings of the purchasing scenario. The new temporal reasoning goals of the shipment scenario and some of the composition goals of the new purchasing scenario required the ability to express conditions that combine arbitrary attributes from the descriptions in arithmetic computations. The ability to express such conditions (called multi-attribute-conditions) has been added to the DSD language and the matchmaking algorithm has been extended to respect these conditions. The arithmetic computations did not pose any problems but the optimal configuration of an offer becomes computationally much harder if multi attribute conditions are involved. If several attributes of an offer that can be configured are connected by a multi attribute condition in the offer (or the corresponding request at hand), the configuration can not be performed locally anymore. In order to find an optimal configuration one would have to iterate over all possible configurations (which is impossible for infinite datatypes like time) or solve an equation system which does not have to be linear in all cases. Our current solution uses a heuristic which guarantees correctness but is not complete. Future research is necessary to carefully restrict the expressivity of the DSD language with regard to multi attribute conditions to be able to find a good balance between expressivity and computational tractability in order to improve on the algorithm that configures an offer in the presence of multi attribute conditions.

The composition capabilities of the DSD matchmaking algorithm are sufficient to address the basic composition requirements of the purchasing scenario. Nevertheless the matchmaking implementation currently does not handle lists well enough to handle Goal C2 of the new scenario. This is the second main direction of future work.

The integration of user preferences within service requests has been a major concern in the design of DSD. DSD provides powerful support for preferences using fuzzy sets in request descriptions. The goals of the new scenario using preferences could be expressed very well.

In contrast, rules are not supported directly by DSD. Therefore some aspects – like the computation of the exptected shipping time in the shipment discovery scenario – had to be delegated to an external service. Since such an external rule evaluation can be smoothly integrated into the DSD matchmaking algorithm via estimation operations it is not planned to add direct rule support to DSD in the near future.

Overall our approach is currently able to solve all but one of the goals in both discovery scenarios. DSD and the supporting DIANE framework have thus been proven to be a powerful tool for the semantic based automation of service oriented computing.

References

1. Küster, U., König-Ries, B., Klein, M.: Discovery and mediation using diane service descriptions. In: First Workshop of the Semantic Web Service Challenge 2006 - Challenge on Automating Web Services Mediation, Choreography and Discovery, Palo Alto, California, USA (2006)
2. Küster, U., König-Ries, B., Klein, M.: Discovery and mediation using diane service descriptions. In: Second Workshop of the Semantic Web Service Challenge 2006 - Challenge on Automating Web Services Mediation, Choreography and Discovery, Budva, Montenegro (2006)
3. Küster, U., König-Ries, B.: Discovery and mediation using diane service descriptions. In: Third Workshop of the Semantic Web Service Challenge 2006 - Challenge on Automating Web Services Mediation, Choreography and Discovery, Athens, GA, USA (2006)
4. Petrie, C.: It's the programming, stupid. IEEE Internet Computing **10** (2006) 95 – 96
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** (2005) 328–364
6. Küster, U., König-Ries, B., Klein, M., Stern, M.: Diane - a matchmaking-centered framework for automated service discovery, composition, binding and invocation. to appear in International Journal of Electronic Commerce (IJEC) - Special Issue on Semantic Matchmaking and Retrieval (2007)
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 4th International Conference on Service Oriented Computing (ICSOC06), Chicago, Illinois, USA (2006)
8. Klein, M., König-Ries, B.: Integrating preferences into service requests to automate service usage. In: First AKT Workshop on Semantic Web Services, Milton Keynes, UK (2004)
9. Küster, U., König-Ries, B., Klein, M., Stern, M.: Diane - an integrated approach to automated service discovery, matchmaking and composition. In: Proceedings of the 16th International World Wide Web Conference (WWW2007), Banff, Alberta, Canada (2007)