SERVICE DISCOVERY WITH SWE-ET AND DIANE
A Comparative Evaluation by Means of Solutions to a Common Scenario

Ulrich Küster[1], Andrea Turati[2], Maciej Zaremba[4], B. König-Ries[1], D. Cerizza[2], E. Della Valle[2]
M. Brambilla[3], S. Ceri[3], F. M. Facca[3] and C. Tziviskou[3]
[1]*Institute for Computer Science, Friedrich Schiller University Jena, 07743 Jena, Germany*
[2]*CEFRIEL, Via Fucini 2, 20133 Milano, Italy*
[3]*Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milano, Italy*
[4]*DERI – National University of Ireland, IDA Industrial Estate, Lower Dangan, Galway, Ireland*

Keywords: Semantic Matchmaking, Service Discovery, Evaluation, Comparison.

Abstract: Semantic service discovery and matchmaking has received increased attention within the last years. Various approaches have been proposed but agreed upon criteria or common use and test cases to objectively evaluate such approaches are widely lacking. In this paper we present an in-depth comparison of the solutions to the discovery problems defined by the SWS-Challenge 2006. By means of this common and independently developed scenario we can develop a much better understanding for the applied technologies in general, but also and in particular for the trade-offs involved in the different approaches.

1 INTRODUCTION

Following the idea of the semantic web and the success of service oriented computing (SOC), the application of semantic technologies to automate central tasks within SOC like service discovery, matchmaking and binding has received increasing attention within the last years. Many frameworks have been proposed and are being actively developed, but up to now there is no consensus about the most appropriate formalism for semantic service description or the best techniques to work with or reason about them. Clearly a unified benchmark would be tremendously helpful in order to compare the pros and cons of the various approaches in an objective way. The Semantic Web Services Challenge[1] (Petrie, 2006) is attempting to create such a benchmark. In a series of three workshops so far, participants have applied their technology to a common set of scenarios and presented their approaches to their peers. This way a good understanding of each others technology could be developed. In this paper we will present a comparative evaluation of the solutions to the SWS-Challenge's discovery scenario, with a focus on the joint solution by the team from Politecnico Milano and CEFRIEL (Brambilla et al., 2006) on the one hand and the one by University Jena (Küster and König-Ries, 2006a; Küster et al., 2006) on the other hand. We will describe the various aspects of these approaches in a structured way and elaborate on the trade-offs involved in each technology. The rest of the paper is organized as follows: In Section 2 we shortly describe the discovery scenario of the challenge. Section 3 contains the detailed comparison of the different solutions to that scenario and Section 4 presents our conclusions.

2 THE SCENARIO

The SWS-Challenge presented two complementary scenarios: the *mediation scenario* integrates a legacy system supporting the RosettaNet protocol, the *discovery scenario* is about how to identify possibly relevant shipment services. In this paper we focus on the latter.

The objective of the discovery scenario is to find the best shipment service, taking into consideration pickup location, delivery location, delivery time, price and similar constraints. Thus the scenario provided implementations of several shipping services, each of them with a WSDL and a more concise natural language description. What makes the scenario

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

Küster U., Turati A., Zaremba M., König-Ries B., Cerizza D., Della Valle E., Brambilla M., Ceri S., M. Facca F. and Tziviskou C. (2007).
SERVICE DISCOVERY WITH SWE-ET AND DIANE - A Comparative Evaluation by Means of Solutions to a Common Scenario.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems*, pages 438-446

Table 1: Comparison between DIANE and SWE-ET.

Feature	DIANE	SWE-ET
Formalism	DE and DSD (custom formalism)	F-logic
Service Descriptions	configurable set of possible effects	WSMO service capabilities
Goal Descriptions	fuzzy set of acceptable effects (fuzziness to express preferences)	WSMO goal capabilities
Matchmaking	set-based: subset value of optimally configured offer in fuzzy request	rule-based: matching rules coded into wgMediator (filter web service instances wrt. given goal)
Negotiation	integrated into matchmaking (negotiable parameters tagged in offers)	integrated into the discovery process with external invocation through WebRatio grounding (negotiable parameters tagged in offers)
Selection	through fuzzy requests (integrated into matchmaking)	done by the user through WebRatio interface
Invocation	automatically done by framework	automatically done by WebRatio grounding

interesting is the realistically heterogeneous nature of the services descriptions, that focus on different characteristics. For example, some specifications are:

- Muller specifies rates on request (cf. invokePrice operation within the WSDL)
- Racer adds 12.50 for each collection order
- Runner requires weight, length and height if package weight exceeds 70 lbs, and states that collection is possible between 1am - 12pm
- Walker applies different rates depending on location, and has some conditions about, for instance, maximum package weight or pickup time
- Weasel ships only within the United States.

Discovery had to be performed by a set of predefined requests with increasing complexity.

3 COMPARISON OF BOTH APPROACHES

In the following we will compare the approaches taken by University Jena as well as Politecnico Milano and CEFRIEL to tackle the discovery problem introduced above. The solution by University Jena is based on its DIANE-framework[2] while the other one is named SWE-ET[3] and combines CEFRIEL's Glue discovery engine[4] with the WebRatio framework[5] from Politecnico Milano.

We adopt a structured approach to compare both solutions along several dimensions. Table 1 shows a compact representation of the comparison result.

[2]http://hnsp.inf-bb.uni-jena.de/DIANE

[3]http://sweet.cefriel.it/

[4]http://glue.cefriel.it/

[5]http://www.webratio.com/

3.1 Formalism Used to Model Ontologies

The goal of the DIANE project is to create a framework that is able to completely automate the whole process of service usage. Thus an ontology language was needed that on the one hand is expressive enough to precisely capture the necessary aspects of service offers and requests but on the other hand is as restricted as possible to ease the matchmaking process and maintain efficient processability. Therefore the approach followed by DIANE is to not use one of the logics commonly employed for semantic service descriptions, but to define its own language specifically tailored towards the use case at hand. Consequently DIANE uses its own ontology language, called DE (*DIANE Elements*) and DSD (*DIANE Service Descriptions*) which has been introduced in (Klein et al., 2005). Ontologies are very lightweight and the description elements of DSD used for ontologies can best be characterized as a small subset of F-logic (Kifer et al., 1995) without rules and quantifiers.

In contrast Glue – the discovery engine used in SWE-ET – is directly based on F-logic. This was motivated by the desire to create a discovery engine that is compliant with WSMO (de Bruijn et al., 2005a). At the time development on Glue started, tools for translating WSML into reasoner-specific formats were missing and WSML itself was a work-in-progress. However, the WSML working group[6] proposed a subset of OWL (named OWL$^-$) that does not fall outside a logic programming framework and allows for a straight-forward combination with rule languages (de Bruijn et al., 2005b). Additionally an expressive datatype support was desired and WSML

[6]http://www.wsmo.org/wsml/

working group showed in (de Bruijn et al., 2004) that OWL-E (Pan, J. Z. and Horrocks, I., 2004) (a proposal for extending OWL with expressive datatype expression) can be added to OWL$^-$, forming a new ontological language named OWL-Flight. Since both OWL$^-$ (Angele and Lausen, 2004) and OWL-Flight can be translated into F-logic, it was decided to implement Glue on a F-logic reasoner.

Thus the Glue approach models ontologies using F-Logic and can benefit from the entailed expressivity: F-logic allows to represent classes, instances, relationships among classes and instances, formulas that use logic operators and quantifiers, rules, and so on. F-logic provides a second-order, object-oriented-style syntax for a first-order logical language. In other words, as described in (Kifer and Lausen, 1989), F-logic has an appearance of a higher-order-logic, but, unlike it, is tractable and has a natural direct first-order semantics. In addition, sound and complete proof procedures for F-logic exist.

To address the SWS-Challenge's scenarios, both teams modeled necessary domain ontologies to capture required concepts like date and time, weight and dimensions, prices, locations, shipment etc. Despite the much bigger expressivity of the full F-logic approach taken by Glue, the modelled ontologies look fairly similar since the current scenario did not require to use complex rules and restrictions in the ontologies.

3.2 Formalism Used to Model Goals and Services

While the underlying ontologies are rather simple, DSD supports more complex and expressive modelling operators to be used in request and offer descriptions. In DSD, service offers are described as the set of effects they can provide wheras service requests are described as the set of effects that are acceptable for the requester. The semantics of DSD defines that one effect out of the request effect set is requested and one effect out of the offer effect set will be provided by a service invocation[7]. Figure 1 shows excerpts of the description of the Muller shipping service in an intuitive graphical notation.

As mentioned, Muller is described by the set of Shipped effects it can provide. Sets are denoted by a diagonal line in the upper left corner of a concept. Generally DSD sets are described by

- a *type condition* to specify the class of admissable instances (like Shipped, Country or Double)

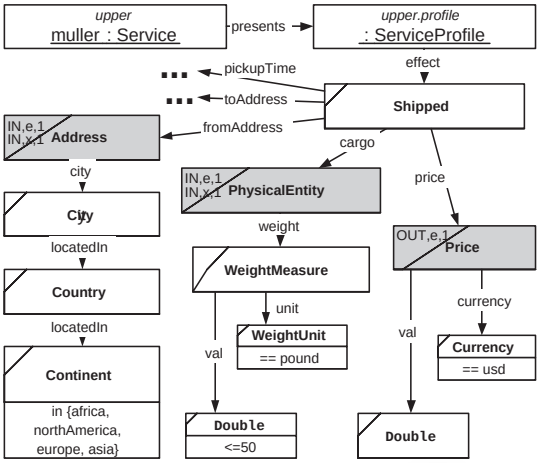

Figure 1: Excerpts from the description of the Muller shipping service. Muller allows shipping within Africa, North America, Europe and Asia and transports only packages up to 50 pounds.

- a number of *direct conditions* to include or exclude named instances (compare with the Continent set) or specify declarative conditions on primitive types (compare with the Double set used for the weight of a cargo)
- a number of *property conditions* that recursively specify the sets of admissable values for the properties of admissable instances (like the *val* and *unit* property of the WeightMeasure set). Property descriptions lead to service descriptions that form trees as shown in Figure 1.

To express preferences, fuzzy sets may be used instead of crisp sets in request descriptions: The higher the set membership value, the higher the preference of the requester. Fuzzy sets are constructed using fuzzy variants of the type and direct condition introduced above together with *connecting strategies* that specify how to compute the membership value of an instance based on the membership values of its properties (A weighted sum might for instance be used to compute the set membership value of a Shipped-instance using the values of its properties cargo, price, pickupTime, ...).

The concept of variables – a special type of set – is used to represent necessary inputs and required outputs of a service invocation. Variables are marked with a gray background in Figure 1. Where allowed by the offer description, variables enable the requester to configure an offer (i.e. influence which of the offered effects will be created). Muller for instance takes the destination address and the cargo to ship as input and provides the price of the shipping operation as output. For more information on DSD, please refer

[7] This may be changed by using special operators, called *iteration directives, see (Küster and König-Ries, 2006a)*

to (Küster and König-Ries, 2006a; Klein et al., 2005).

In Glue, web services and goal descriptions are represented in F-logic, like ontologies. To model the requests a shipping goal class was designed, capturing the desired capabilities as post-conditions following the WSMO modeling approach (de Bruijn et al., 2005a). Likewise the semantics of the offer descriptions were captured by a web service class for shipment. The restrictions that must hold in order to invoke a service were modelled as assumption and the result provided by an invocation as post-conditions. As explained in (Della Valle and Cerizza, 2005), Glue refines the WSMO discovery conceptual model by making the notion of class of goals and class of web service description explicit and by making a clear separation between instances and classes of goals and web services.

An example of the F-logic description of the web service instance for the Muller shipping service can be found in section 3.4.

In a service description there could be properties that need to be negotiated, in order to assume a specific value. Properties that require invocation of external web services to gather more information are managed in a special way. They are annotated with special tags, so that Glue can invoke WebRatio when it recognizes these tags and WebRatio can negotiate them. This aspect is detailed in section 3.4.

Not surprisingly, F-logic was sufficiently expressive to capture all details of the given offer and goal descriptions. However F-logic is not a temporal logic, therefore a date-time ontology (including concepts like distance between two different instants of time) has been modeled in order to have the possibility to manage temporal constraints for the scenario.

Glue needs to exploit the notion of current time given by WebRatio. When a goal description is inserted by means of the graphical interface of WebRatio, the value of the current time is passed from WebRatio to Glue together with the goal description so that Glue can use it during the matchmaking.

In DIANE, since DSD classes and instances are internally represented as Java classes, custom Java code may be injected into instances and classes. In order to deal with temporal aspects, DIANE could exploit a special time instance *now* that could access the System time to assert the actual date and time. On the other hand, the lack of rules and arithmetic expressions in DSD were hindering the modeling of some aspects of offer and goal descriptions. Rule-based price computations (e.g. different prices based upon the delivery continent) could not be expressed in DSD. As a workaround for that problem the computation of the price was delegated to an external web service similar to what was anyway required by one out of the five available offers (see Section 3.4). Beside that, DSD currently also does not support the expression of arithmetic computations[8] as needed for instance to model that a shipping price is made up of a constanc factor multiplied with the weight of a package. The workaround for the problem is the same.

Even though finally all aspects could be captured in both SWE-ET or DIANE, the greater expressivity of the F-logic approach taken by Glue proved advantageous in the above mentioned cases.

3.3 The Process of Matchmaking

Reasoning: Since DSD does not build on any common logic formalism it cannot benefit from standard reasoning tools. Instead a special custom reasoning operation *subset* has been defined that solves the problem of service matchmaking.

For a list of given DSD offer descriptions O and a given DSD request r, a matchmaker has to answer two questions for each $o \in O$: What is the subset value of o's effect sets in r's fuzzy effect sets (how well is o contained in what r requests) and which configuration of o yields the best such value? Thus $subset(O,r)$ returns a list of altered offers O', sorted by the fuzzy subset value of each $o'_i \in O'$ wrt. r (called *match value*) and each $o'_i \in O'$ corresponds to exactly one $o_j \in O$ where o'_i differs from o_j exactly by the fact that all input variables in o_j have been filled with a concrete instance value.

The current Java based implementation of subset steps through the graphs of each o_i and r synchronously in order to calculate the matching value in [0,1] as well as the optimal filling of all input variables. It is worth mentioning that thus the DSD matcher does not only passively select the most appropriate offer but also actively configures and optimize each offer where possible. The expressivity of DSD has been tailored with the goal to support efficient computability of *subset*. For further information on matchmaking of DSD descriptions please refer to (Klein et al., 2005).

In contrast to DSD, which defines a custom reasoning operation, Glue can build upon standard reasoning tools for F-logic. Flora-2[9] – a plug-in of the XSB inference engine[10] based on Prolog – was chosen as inference engine. The language of Flora-2 is a dialect of F-logic with numerous extensions, including HiLog and Transaction Logic. In Glue, discovery returns all the web services descriptions that match

[8] This is currently work in progress

[9] http://flora.sourceforge.net/

[10] http://xsb.sourceforge.net/

the request at predefined levels. The level is computed by evaluating a wgMediator (a WSMO entity that is in charge of mediating between web service and goal), that basically specifies a set of F-logic rules specific to match instances of particular web services classes with an instance of a particular goal class. Given a goal instance, in order to identify all instances of the web services classes that match it, each rule is applied on a web service description at a time, resulting in a value that states whether the rule is satisfied or not. Depending on what rules are satisfied, a discrete value stating the level of match is returned.

For example, in the wgMediator for shipment four level of match were defined. The last one is the level associated to the least accurate match, in which only the following rules must be satisfied:

- the weight of the good declared in the goal must be lower than the maximum that the service accepts for shipping
- pickup and delivery locations requested in the goal must be covered by the service.

Execution framework and semantics: DSD assumes a common ontology used by all participants of the matchmaking process. A middleware framework is supplied to support service usage. During setup the common domain ontologies need to be deployed at the middleware. Afterwards, offer descriptions (specified in DSD and based on those ontologies) can be published to a repository within the middleware. Requests as well as offers are internally represented as Java objects. A request agent handles the automatic execution of requests that are sent to the middleware. It causes the matcher agent to match the request against the available offers (see above), picks the best matching offer, forwards it to an invocation agent and takes care that the outputs of the service invocation are sent back to the requester.

Glue embraces the notion of mediation as specified in (de Bruijn et al., 2005a). Every element involved in the discovery process is inserted in the reasoner as a F-logic description. During discovery, the reasoner performs inferences upon these descriptions. The result of the inference process represents the result of the discovery. In particular, during *setup time* the F-logic descriptions of web services and goal classes are inserted in Glue, as well as the F-logic descriptions of both domain ontologies and all the required mediators. At *publishing time*, provider entities publish instances of web service descriptions simply by referring to the correct class of web service description and providing values to all the necessary parameters. Then, at *discovery time*, when a user formalizes a goal instance, the goal class the goal instance belongs to is selected. By using a set of appropriated ggMediators, other semantically equivalent goals are identified by translating the original goal. Then, for each goal previously identified, Glue takes the web service class that match it by using a wgMediator, which connects a goal class to a web service class. In other words, the instances of web services descriptions can be roughly filtered on the basis of the target of the detected wgMediators. Finally, a matchmaking between the goal instance description and the web services instances descriptions is performed and then the user can select (and invoke) one of the remaining web services descriptions in order to satisfy the goal.

Selection: Regarding selection the views taken by SWE-ET and DIANE are quite different. In the SWE-ET framework, discovery and selection are viewed as separate tasks. Glue as a discovery engine is able to discover a set of web services that can satisfy a specified request, which is represented by a goal. Selection is viewed as an additional step that follows the discovery and is responsible for choosing a single web service to be invoked, starting from the set of web services returned by discovery. For this reason, Glue does not include selection. In the Glue approach, selection has to be left in charge of the user, which is the only entity that can take such a decision. Glue only supports the user's decision by applying ranking of the results corresponding to the above mentioned match levels. Glue does so by assigning a value to each matching rules that represents the importance of that rules.

In contrast DIANE is aiming at completely automating the whole process of service usage. This is only possible if selection is performed by the matchmaking process, too. This in turn is feasible only if it is possible to precisely capture user preferences within service requests and efficiently use that additional information during matchmaking. The first is achieved through DSD's fuzzy elements, the latter through the application of the specifically tailored subset operation used for matchmaking. Thus DSD is able to provide a more finegrained matching compared to Glue at the price of restricted expressivity and limited compatibility to other semantic service frameworks.

3.4 Dynamic Aspects of Service Descriptions

Section 3.2 shows how to model static aspects of a web service description, assuming no dynamic dependencies. However, one shipping service (Muller)

required to inquire the price of a shipping operation dynamically by calling a particular web service endpoint.

To cope with such requirements, DSD supports a simple choreography to interact with services where an arbitrary number of *estimation steps* is followed by a single execution step. Estimation steps must not have effects on the real world and can be used to gather dynamical information from a service provider. In Figure 1, Muller declares that given the shipping address and the cargo properties as input, the price of a shipping operation can be retrieved as output of the first estimation step (denoted by markers `OUT,e,1` and `IN,e,1`)[11].

Thus – if necessary – the matcher will initiate a call of the associated operation and dynamically complement Muller's description corresponding to the retrieved information. This procedure was flexible enough to support all dynamic aspects contained in the scenario. Additionally it was used to delegate arithmetic computations as well as the evaluation of rules needed to compute the price for a shipping to an external web service to overcome some of the expressivity limits of DSD (compare to Section 3.2).

Originally Glue had not been able to deal with dynamic aspects. In order to overcome this limitation, Glue has been extended by making a clear separation among:

- *discovery capabilities*, which represent the static description of the service
- *negotiation capabilities*, which represent the dynamic description of the service that needs to be evaluated by invoking it
- *selection capabilities*, which include non-functional descriptions.

This extension has had a minimal impact on the SWE-ET infrastructure. It has been sufficient to add new features to the execution semantics at the end of the entire discovery process, in order to perform the negotiation.

When a web service is published in Glue, in the case that the service includes special parameters that need to be negotiated (e.g. shipping price), its description has to be annotated with special tags, which point out what parameters have to be negotiated. For example, the following code shows how the *price* of the Muller service has been modeled in order to enable negotiation. In particular, with respect to the code shown below, a tag stating that it is necessary to invoke the *invokePrice* operation of the Muller service in order to negotiate the values of the price has been added. In addition, the request message (*invokePriceRequest*) that has to be sent to start the negotiation has been added to Muller's description.

```
wsdInstance_Shipment11:wsdClass_Shipment[
  nonFunctionalProperties->_#[
    dc_title->'Muller Shipment Service'
    ...  ],
  capability->_#:capabilityWSD_Shipment[
    assumption->_#:
    restrictionsOnShipmentService[
        maximalGoodWeight->50,
        ...    ],
    postcondition->_#:
           providesShipmentService[
      pickupLocations->>{africa, ...},
      price->>{
        _#:shipmentPricing[
          basePrice->0,
      //negotiate_operation:
      //Muller/invokePrice(out Price)
          pricePerWeight->0,
          additionalPricePerCollect->(-1)
        ]  ...
].
invokePriceRequest[
  country=>location,
  packageInformation[
    weight=>integer,
    lenth=>integer,
    height=>integer,
    width=>integer
  ]
].
```

At the first step of the discovery, Glue deals with the static descriptions of services only. When it identifies a service description in which there are some parameters that have been annotated with a special tag (i.e. the *negotiate_operation* tag in the previous example), Glue starts a negotiation by delegating it to WebRatio. In other words, Glue is responsible for modeling a service description (including also its negotiation capabilities) and starting the negotiation process when it is necessary to dynamically get the value of a parameter. WebRatio is responsible for handling the actual invocation (including the grounding toward SOAP messages). After negotiation, WebRatio returns the actual value for the parameter, so that Glue can update the service description adding this value. Finally, Glue evaluates whether the updated instance of service satisfies the goal by applying the appropriate rules and accepts or rejects the service correspondingly.

[11] Details how to execute that estimation step are specified in an offer's grounding, which is beyond the scope of this paper. Please refer to (Küster and König-Ries, 2006b) for further information.

3.5 Invocation

Automated invocation of offers is directly supported by DIANE. In case of the before mentioned estimation steps the corresponding invocations can be initiated by the matcher directly and can be performed interweaved with the matchmaking process. Regarding the final execution of the service, the matcher – as outlined in Section 3.3 – outputs a list of readily configured offers, i.e. offers where all necessary input values have been set. The remaining task performed by the invocation agent is to perform the necessary lowering to create an appropriate XML message to send to the offer implementation and perform lifting on the returned response message. This is done using simple declarative mapping rules that map between DSD concepts and XML data (see (Küster and König-Ries, 2006b) for details).

In the SWE-ET approach matchmaking and invocation are performed using different technologies. Invocation of a web service is not directly executed by Glue, but is left to the application in which Glue is integrated – in the case of phase-III of the SWS-Challenge an external invocation component implemented within the WebRatio framework. This approach is described in (Zaremba et al., 2007).

3.6 Comparison with Other Solutions

DERI[12] has provided a solution that is quite similar to the one of CEFRIEL-Politecnico di Milano. In particular both are based on the same framework for Semantic Web services - WSMO. In this section we emphasize the main differences between the two solutions.

The solution by DERI is based upon WSMX – the WSMO execution environment – and thus natively supports WSML, in the contrary to Glue where WSMO is encoded in F-logic by means of Flora-2 syntax. Semantic Web services, goals and ontologies have been modeled directly in WSMT[13], which provides versatile support for modeling WSMO elements.

Since logical rules have been used quite extensively in order to explicitly describe various shipping criteria of different shippers, DERI expresses the rules by means of WSML-Rule flavor of WSML, while CEFRIEL expressed them in the same formalism used also for goals and services descriptions. Instead of Flora-2, DERI used KAON2[14] interfaced via the WSML2Reasoner[15] framework as internal reasoner to inference over the provided functionality of a service. KAON2 comes with support for elaborate rules and arithmetic expressions, therefore the WSMX Discovery component did not have to resort to an external arithmetic services but required calculations were carried out internally within the context of reasoner.

Similarly to Flora-2, KAON2 is a high precision reasoner giving explicit responses without the direct support for fuzziness or preferences. DERI has provided simple support for the preferences via non-functional properties within a goal which specify which criteria should be taken into the account when selecting the most suitable service.

The DERI submission directly supports service contracting as necessary in the case of Muller where the shipping price had to be dynamically obtained. WSMO Choreography expressed in ontologized Abstract State Machines (snippet below) has been used for specifying the necessary interaction with the service. This contracting Choreography is utilized during the discovery phase and its result (i.e. price in this case) is integrated into the reasoning context. Once the service has been selected for the execution phase there is a separate Choreography allowing to consume service functionality.

```
choreography contracting#choreogr
 stateSignature
 in mu#pQuoteReq withGrounding { ... }
 out mu#pQuoteResp withGrounding { ... }
forall {?pQuoteReq} with (
 ?pRequest memberOf mu#pQuoteReq) do
    add( # memberOf mu#pQuoteResp)
endForall
```

WSMX uses a dedicated Communication Manager component for handling the external communication with service requesters and with involved services. The Communication Manager manages adapters which are responsible to perform the necessary lifting and lowering (i.e. translate between XML and WSML). Lifting is required when moving from syntactic data representation to a rich ontology model, while lowering does the opposite, namely mapping downwards to non-semantic data models (e.g. XML, EDI, RosettaNet, others).

4 CONCLUSIONS

Overall the discovery performed by SWE-ET on the one hand and DIANE on the other is quite different.

[12] http://www.deri.org
[13] http://wsmt.sourceforge.net
[14] http://kaon2.semanticweb.org
[15] http://tools.deri.org/wsml2reasoner

DIANE uses a fuzzy set based approach to matchmaking. Offers are modelled as the configurable set of effects a service can provide wheras requests are modelled as the fuzzy set of similar effects a user is willing to accept. This maximizes the chance to find a match and allows to integrate user preferences, thus providing a very finegrained ranking of matching offers. But to achieve this and still maintain efficient computability the expressivity of the employed language (DSD) had to be restricted quite strictly. An example will be given below.

Glue in contrast uses a very different matchmaking philosophy. Glue performs matchmaking based on the notion of a single goal instance and the available web service instances. Matchmaking is then performed by evaluating rules that check whether a particular web service instance under consideration is suitable for the particular goal at hand. In principle this eases the matchmaking since it is easier to compare two instances than to compare a fuzzy with a configurable set. Thus – in turn – Glue is able to support a much more expressive language (F-logic) without compromising efficient computability.

To illustrate the trade-offs, assume the following two examples: A shipper supports collection of packages on Monday through Friday, the preferred day can be specified within the ordering process. A requester accepts collection on either Friday, Saturday or Sunday. Using DIANE's set based approach, this could be directly encoded and the matcher would correctly detect a match and automatically choose Friday as the proper collection day. In the SWE-ET approach this could not be captured directly (since it requires set-based matching).

Now assume the requester had required a particular collection day instead of allowing multiple options but the conditions of the shipping (regarding price, available collection times, ...) vary depending on the chosen collection day. This could be easily modelled correctly in SWE-ET but not directly expressed using DSD (lack of expressivity regarding rules).

It should be mentioned however, that in both cases certain workarounds allow the respectively inferior approach to deal with the issue at hand. This is also reflected by the fact that both approaches were successfully applied to solve the SWS-Challenge discovery scenarios released so far.

REFERENCES

Angele, J. and Lausen, G. (2004). Ontologies in F-logic. In *Handbook on Ontologies*, pages 29–50.

Brambilla, M., Celino, I., Ceri, S., Cerizza, D., della Valle, E., Facca, F., and Tziviskou, C. (2006). 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 Semantic Web Service Challenge 2006*, Athens, GA, USA.

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., and Stollberg, M. (2005a). Web service modeling ontology (wsmo). W3C Member Submission 3 June 2005.

de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2004). D20.3 OWL flight. Technical report, WSML.

de Bruijn, J., Polleres, A., Lara, R., and Fensel, D. (2005b). D20.1 OWL$^-$. Technical report, WSML.

Della Valle, E. and Cerizza, D. (2005). The mediators centric approach to automatic web service discovery of glue. In *MEDIATE2005*, volume 168 of *CEUR Workshop Proceedings*, pages 35–50. CEUR-WS.org.

Kifer, M. and Lausen, G. (1989). F-logic: A higher-order language for reasoning about objects, inheritance, and scheme. In *Proc. ACM SIGMOD Conf.*, page 134, Portland, OR.

Kifer, M., Lausen, G., and Wu, J. (1995). Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4):741–843.

Klein, M., König-Ries, B., and Müssig, M. (2005). What is needed for semantic service descriptions - a proposal for suitable language constructs. *International Journal on Web and Grid Services (IJWGS)*, 1(3/4):328–364.

Küster, U. and König-Ries, B. (2006a). Discovery and mediation using diane service descriptions. In *Third Workshop of the Semantic Web Service Challenge 2006*, Athens, GA, USA.

Küster, U. and König-Ries, B. (2006b). Dynamic binding for BPEL processes - a lightweight approach fo integrate semantics into web services. In *Second International Workshop on Engineering Service-Oriented Applications: Design and Composition (WESOA06) at ICSOC06*, Chicago, Illinois, USA.

Küster, U., König-Ries, B., and Klein, M. (2006). Discovery and mediation using diane service descriptions. In *Second Workshop of the Semantic Web Service Challenge 2006*, Budva, Montenegro.

Pan, J. Z. and Horrocks, I. (2004). OWL-E: Extending owl with expressive datatype expressions. Technical report, IMG/2004/KR-SW-01/v1.0, Victoria University of Manchester.

Petrie, C. (2006). It's the programming, stupid. *IEEE Internet Computing*, 10(3):96, 95.

Zaremba, M., Vitvar, T., Moran, M., Brambilla, M., Ceri, S., Cerizza, D., Valle, E. D., Facca, F. M., and Tziviskou, C. (2007). Towards semantic interoperabilty: In-depth comparison of two approaches to solve mediation tasks. In *Comparative Evaluation of Semantic Web Service Frameworks Special Session at ICEIS 2007*.