Surfing the Service Web

Sudhir Agarwal[1], Siegfried Handschuh[1], and Steffen Staab[1,2]

[1] AIFB, University of Karlsruhe
{agarwal,sha,sst}@aifb.uni-karlsruhe.de,
http://www.aifb.uni-karlsruhe.de/WBS
[2] Ontoprise GmbH, 76131 Karlsruhe, Germany,
http://www.ontoprise.com/

Abstract. The way that web services are currently being developed places them beside rather than within the existing World Wide Web. In this paper we present an approach that combines the strength of the World Wide Web, viz. interlinked HTML pages for presentation and human consumption, with the strength of semantic web services, viz. support for semi-automatic composition and invocation of web services that have semantically heterogeneous descriptions. The objective we aim at eventually is that a human user can seamlessly surf the existing World Wide Web and the emerging web services and that he can easily compose and invoke Web services on the fly without being a software engineer. This paper presents our framework, OntoMat-Service, which trades off between having a reasonably easy to use interface for web services and the complexity of web service workflows. It is not our objective that everybody can produce arbitrarily complex workflows of web services with our tool, the OntoMat-Service-Surfer. However, OntoMat-Service aims at a service web, where simple service flows are easily possible — even for the persons with not much technical background, while still allowing for difficult flows for the expert engineer.

1 Introduction

The Stencil Group defines web services as: loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard internet protocols. Though this definition captures the broad understanding of what web services are, it raises the question, what web services have to do with the web. Even if HTTP is used as a communication protocol and XML/SOAP to carry some syntax this appears to be a rather random decision than a deeply meaningful design.

We believe that it makes sense to actually integrate the strengths of the conventional World Wide Web, viz. lightweight access to information in a highly-distributed setting, with the strengths of web wervices, viz. execution of functionality by lightweight protocols in a highly-distributed setting. To seamlessly integrate the two aspects we envision a *service web* that uses XHTML/XML/RDF to transport information and a web service framework to invoke operations and a

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 211–226, 2003.

framework, OntoMat-Service, to bind the two aspects together. OntoMat-Service offers an infrastructure, OntoMat-Service-Surfer, that allows

- for seamlessly browsing conventional web pages, including XHTML advertisements for web services;
- for direct, manual invocation of an advertised web service as a one-off use of the service;
- for tying web service advertisements to each other when browsing them;
- for tying web service advertisements to one's own conceptualization of the web space when browsing them; and
- for invoking such aggregated web services.

For these objectives, we build on numerous existing technologies like RDF [9], ontologies [2] or WSDL [1]. To integrate the web and web services into the service web, we make specific use of a new type of *semantic annotation* [6], namely *deep annotation* [7].

The paper proceeds as follows. We first describe a simple use case for OntoMat-Service (cf. section 2), including a detailed WSDL description of a web service used for the running example. In section 3, we describe the process that allows to turn web services into a service web and that lets a user surfing the web with OntoMat-Service-Surfer exploit the very same tool to aggregate and invoke web services. The first step of this process, i.e. advertising web services in a form that combines presentation for human and machine agent consumption, is sketched in section 4. The second step of this process, i.e. using browsing and semantic deep annotation to tie together conceptual descriptions, is described in section 5. The third step comprises the generation of simple web service flows and is described in section 6. The fourth and final step described in section 7 deals with the invocation of web service flows. Before we conclude, we overview some related work.

2 Use Case

A typical use case supported by OntoMat-Service is the following (adapted from a larger scenario in [11]): An employee in a small enterprise needs a new laptop. In order to buy one he defines the characteristics of the laptop like processor speed, disk size, etc. Based on the configuration of the laptop he collects offers from laptop vendors. When he receives an offer he also solicits insurance terms from a third party. Once the most reasonable laptop and the best insurance contract terms are determined, the employee purchases the laptop and closes the service contract.

In our scenario, we assume a laptop vendor and an insurer offering web services with two operations each, i.e. `getLaptopOffer` / `buyLaptop` and `getInsurance Terms` / `closeServiceContract`, respectively. The sequence of operations that must be executed by the customer is depicted in Figure 1.

The laptop vendor and the insurer being web service providers describe their web services with WSDL documents. In Figure 2, we show

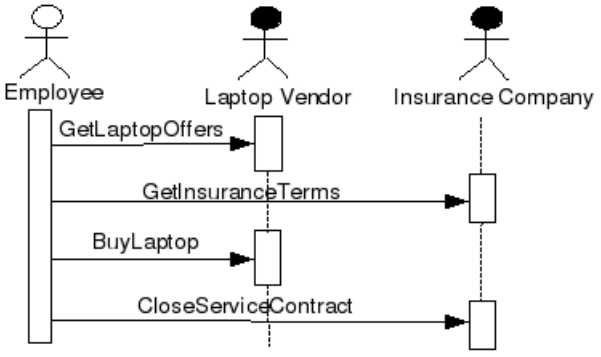

Fig. 1. Sequence Diagram for the Use Case

```
<?xml version="1.0" encoding="UTF-8"?> <definitions
name="LaptopService"
    targetNamespace="http://laptop.wsdl/laptop/"
    <types>
       <rdf:RDF>
          <rdfs:Class rdf:ID="Laptop">
             <rdfs:label>Laptop</rdfs:label>
          </rdfs:Class>
          <rdf:Property rdf:ID="diskSpace">
             <rdfs:label>diskSpace</rdfs:label>
             <rdfs:range rdf:resource="&rdfs;Literal"/>
             <rdfs:domain rdf:resource="#Laptop"/>
          </rdf:Property>
           ...
          <rdf:Property rdf:ID="price">
             <rdfs:label>price</rdfs:label>
             <rdfs:range rdf:resource="&rdfs;Literal"/>
             <rdfs:domain rdf:resource="#Laptop"/>
          </rdf:Property>
           ...
       </rdf:RDF>
    </types>
    <message name="getOffersRequest">
        <part name="processorSpeed" type="rdf:ID=processorSpeed"/>
        <part name="diskSpace" type="rdf:ID=diskSpace"/>
    </message>
    <message name="getOffersResponse">
        <part name="laptopOffers" type="rdf:ID=laptops"/>
    </message>
        ...
    <portType name="LaptopService">
        <operation name="getLaptopOffers" parameterOrder="processorSpeed diskSpace">
            <input message="tns:getOffersRequest" name="getOffersRequest"/>
            <output message="tns:getOffersResponse" name="getOffersResponse"/>
        </operation>
        ...
    </portType>
</definitions>
```

Fig. 2. Web Service Description of Laptop Vendor

```
@prefix rdfs: <http://www.w3.org/rdf-schema#>. @prefix : <#>.
@prefix a rdf:Type.
```

Fig. 3. N3 shortcuts

how a conventional WSDL document of the laptop vendor located at `http://laptop-vendor.de/lap top.wsdl` might look like.[1]

The WSDL document describes:

- *Data type definitions* in the XML element `types`. They are only sketched in figure 2 as they correspond to the laptop vendor's ontology depicted in N3[2] in figure 4. Thereby, we assume the definitions given in Figure 3. In our running example, the WSDL document of the laptop vendor, we describe the class Laptop.
- *Messages* that a service sends and/or receives and that constitute the web service operations in the XML element `portType`. For instance, our running example specifies '`getOffersRequest`' that a potential customer would send to the laptop vendor to solicit an offer. `getOffersRequest` must be provided with two arguments, namely processor speed and disk size. It returns a set of laptop offers with properties such as specified in the vendor ontology (cf. WSDL document in Figure 2 and vendor ontology in Figure 4).

WSDL provides a naming convention for URIs such that each conceptual element (e.g., `types`, `portType`, etc.) of a WSDL document can be uniquely referenced. Such a URI consists of a `targetNamespace` pointing to the location of the WSDL document and to element names of the WSDL document. For example, the URI `http://laptop.wsdl/laptop/#part(getOffersRequest/diskSpace)` refers to the second part (`diskSpace`) of the message `getOffersRequest` of the WSDL document in Figure 2 (cf. [1] for further specifications).

The web service description of the insurer looks similarly. We here only mention that the insurer provides the operations `getInsuranceTerms` and `closeServiceContract`. `getInsuranceTerms` requires a description of Laptop (according to the insurer's ontology in Figure 5) and a timePeriod, for which the contract is supposed to run. `getInsuranceTerms` returns a set of insurance terms available.

In the remainder of the paper, we assume that the customer has the plan depicted in Figure 1. However, in our running example, we will mostly focus on the first two steps to illustrate our framework.

[1] The single ideosyncrasy we have here is that the WSDL document employs RDFS in order to describe the data structures instead of the more common XML schema — though actually WSDL does not require XML Schema and it allows RDFS.

[2] Notation 3 or N3 is basically equivalent to RDF in its XML syntax, but more compact. Cf. http://www.w3.org/DesignIssues/Notation3

```
:Laptop a rdfs:Class.
:price rdfs:domain :Laptop;
       rdfs:range :rdfs:Literal.
:diskSpace rdfs:domain :Laptop;
           rdfs:range :rdfs:Literal.
:processorSpeed rdfs:domain :Laptop;
                rdfs:range :rdfs:Literal.
:laptopID rdfs:domain :Laptop;
          rdfs:range :rdfs:Literal.

:Offer a rdfs:Class.
:laptops rdfs:domain :Offer;
         rdfs:range :Laptop.

:Sale a rdfs:Class.
:laptop rdfs:domain :Sale;
        rdfs:range :Laptop.
:creditCardNumber rdfs:domain :Sale;
                  rdfs:range :Literal.
:customerReceipt rdfs:domain :Sale;
                 rdfs:range :Literal.
```

Fig. 4. Ontology of the laptop vendor

```
:Laptop a rdfs:Class.
:id rdfs:domain :Laptop;
    rdfs:range :Literal.

:ContractTerms a rdfs:Class.
:laptop rdfs:domain :ContractTerms;
        rdfs:range :Laptop.
:timePeriod rdfs:domain :ContractTerms;
            rdfs:range :Literal.
:price rdfs:domain :ContractTerms;
       rdfs:range :Literal.
```

Fig. 5. Ontology of the insurance company

```
:Product a rdfs:Class.
:id rdfs:domain :Product; rdfs:range :Literal.

:HardDisk a :Product.
:diskSize rdfs:domain :HardDisk; rdfs:range :Literal.
:Computer a :Product.
:hasHDD rdfs:domain :Computer; rdfs:range :HardDisk.
:price rdfs:domain :Computer; rdfs:range :Literal.
:cpuSpeed rdfs:domain :Computer; rdfs:range :Literal.

:Agent a :rdfs:Class.
:Company a :Agent.
:creditCardNumber rdfs:domain :Company; rdfs:range :Literal.

:Purchase a :rdfs:Class.
:hasBuyer rdfs:domain :Purchase; rdfs:range :Agent.
:hasObject rdfs:domain :Purchase; rdfs:range :Product.

:Insurance a :rdfs:Class.
:hasObject rdfs:domain :Insurance; rdfs:range :Product.
:price rdfs:domain :Insurance; rdfs:range :Literal.
:timePeriod rdfs:domain :Insurance; rdfs:range :Literal.
```

Fig. 6. Ontology of the customer

3 Overview of the Complete Process of OntoMat-Service

Figure 7 shows the complete process of our framework, OntoMat-Service. First, the figure consists of process steps, which are illustrated by a circle representing the step and a person icon representing the logical role of the person who executes the step, viz. service provider, annotating Service Web surfer and a user invoking a Web Service. The two latter roles typically coincide. Second, the figure comprises information that is used by a person or by OntoMat-Service-Surfer in a process step.

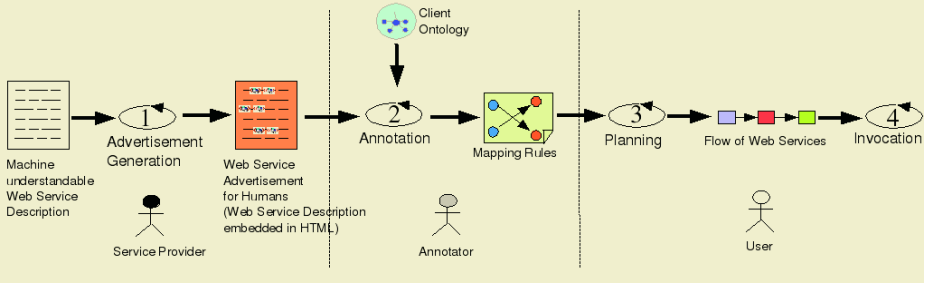

Fig. 7. The Complete Process of OntoMat-Service

The four main steps run as follows:

Init: OntoMat-Service starts with a common WSDL web service description by the service provider (e.g., Figure 2). Obviously, the WSDL document is primarily intended for use by a machine agent or a software engineer who has experience with web services. It is not adequate for presenting it to a user who is 'only' expert in a domain.

Web Service Presentation (Step 1): In the first step, the web service provider makes the web service presentation readable as a nicely formatted (X)HTML document — possibly including advertisements, cross-links to other HTML pages or services, or other items that make the web page attractive to the potential customer (cf. Section 4 for details).

Thereby, it is important that the understandable, but informal description of the web service is implicitly annotated to relate the textual descriptions to their corresponding semantic descriptions in their WSDL document.

Step 1 is a manual step that may be supported by tools such as *WSDL Documentation Generator* from *http://www.xmlspy.com*. However, we would not assume that tools like *WSDL Documentation Generator* would be sufficient to generate an amenable presentation, as they still produce rather rigid and technically oriented descriptions.

Result 1: Human understandable web page that advertises the web service and embeds/refers to machine understandable web service descriptions (WSDL + ontology).

Deep Annotation (Step 2): At a client side, a potential user of the web service browses the web page. OntoMat-Service-Surfer shows the web page like a conventional browser. In addition, OntoMat-Service-Surfer highlights human-understandable items (e.g. text phrases) that associate an underlying machine-understandable semantics.

The logical role of the user here is one of an annotator/surfer. He can decide to just view the page and proceed directly to step 4 (described below). Alternatively, he can decide to map some of the terminology used in the web page of the web service to his own terminology (or to the terminology of someone else).

For the latter purpose, he loads an ontology into OntoMat-Service-Surfer (if it is not already pre-loaded). Then he aligns terminology mentioned in the web page by drag'n'drop-ping it onto the ontology loaded into OntoMat-Service-Surfer. OntoMat-Service-Surfer generates mapping rules from these annotations that bridge between the ontology of the service provider and the ontology loaded into OntoMat-Service-Surfer (cf. Section 5 for details).
Typically, the user will map to more than one web service, i.e. often he will map to different ontologies.

Result 2: Sets of mapping rules between web service ontologies and pre-loaded ontology.

Web Service Planning (Step 3): At the client side, a user might view the web services as well as their annotations that yield mapping rules. The third logical role here is one of a service planner and invocator (this logical role is shared between the third and fourth step). For this purpose, the user decides to select

- a set of web service operations he wants to use and
- a set of mapping rules he wants to use.

The reader may note that very frequently the roles of an annotator/surfer and a service invocator will just coincide. Hence, the two selections just mention will take place implicitly — just by the web service pages he has browsed and the annotations that the service invocator has performed in step 2 of the OntoMat-Service process.
Once the two selections have been performed im- or explicitly, a module for web service planning will compute logically possible web service flows. For this objective, web service planning may employ a rich set of knowledge: goals, pre-conditions of web services, post-conditions of web services, previous similar cases, etc. In the current version of OntoMat-Service we just exploit the pre- and post-conditions derived from mapping one web service output to another web service input via the customer ontology. The web service description in the associated WSDL document describes what types are required for the input of a web service and what types appear in the output of a web service. Since data that wanders from one web service to the next can only proceed if types are compatible, OntoMat-Service-Surfer can compute a restricted set of possible web service flows (cf. Section 6).
Though in general this model may be too weak to compute complex flows it is quite sufficient and straightforward to use with a small number of selected and semantically aligned web services — such as an end user or prototype builder will use.

Result 3: Sets of possible web service flows.

Web Service Invocation (Step 4): The final user, i.e. the invocator, can select one such flow from the list or modify any, if none of them fits his needs. Obviously, he can always create a new flow on his own. Once the user has a flow, that fulfils his current needs, he invocates the flow (cf. Section 7). During the execution, the tranformation of the data of one ontology to another will happen automatically via the mapping rules. The user achieves his goal at the completion of the invocation of the web service flow.

```
<html><head><title>Laptop Vendor Service</title></head> <body><h1
align="center">Laptop Vendor Service</h1> <p><h2>getLaptopOffers</h2>
This service delivers the top offers of the laptops available in
the city. We have the largest archive of the laptop offers for the
city. So, the possibility that you find your desired laptop at a
reasonable price is very high. Just try it and get convinced from
our great offers. <ul><li> <span
wsdlLocation="http://laptop-vendor.de/laptop.wsdl"
elementURI="http://laptop.wsdl/laptop/#part(getOffersRequest/processorSpeed)">
<b>Processor speed</b> </span> Specifies the speed of the
processor. Please use only the units "MHz" and "GHz". For example,
"2GHz", "1.4GHz" and "1600MHz" are valid whereas "1800" or
"170000KHz" are invalid. </li><li> <span
wsdlLocation="http://laptop-vendor.de/laptop.wsdl"
elementURI="http://laptop.wsdl/laptop/#part(getOffersRequest/diskSpace)">
<b>Disk space</b> </span> Specifies the disk space. Please use
only the units "GB" and "MB". For example, "20GB", "30.5GB" are
valid whereas "40" or "25000KB" are invalid. </li><li> <span
wsdlLocation="http://laptop-vendor.de/laptop.wsdl"
elementURI="http://laptop-vendor.wsdl/laptop/#part(getOffersResponse/laptopOffers)">
<b>Top Offers</b> </span> This is the list of the most reasonable
offers available in the city that fulfill your requirements.
</li></ul></p>
                    ...
</body></html>
```

Fig. 8. Web Service Description as HTML Page

4 Semantic Web Page Markup for Web Services

In this section we show how a web service provider can semantically annotate the web pages describing his web services. Such a combined presentation allows for improved ways to find the web services (e.g., by a combined syntactic/semantic search engine) and it enables a user to define mapping rules between the ontology used in the web service description and the client's ontology.

The basic idea is that a conventional HTML page about the web service and web service parameters is extended by URIs referring to conceptual elements of the corresponding WSDL documents. To carry these two pieces of information, we use `wsdlLocation` and `elementURI` inside the `span` tags. In Figure 8, we show how such an web service advertisement(HTML page) for the laptop vendor service might look like.

When such an HTML page is opened in OntoMat-Service-Surfer, the `span` tags are interpreted and elements between `<span>` and `</span>` are highlighted to support the annotation step described in the next section.

5 Browsing and Deep Annotation

In this section, we describe the second main step of the OntoMat-Service process. This step consists of browsing web pages about web services with OntoMat-Service-Surfer. Thereby, the user may deep-annotate [7] these web pages generating mapping rules between a client ontology and the ontologies referred to in the WSDL documents as a 'side effect'. We call this action 'deep-annotation' as its purpose is not to provide semantic annotation about the surface of what is

being annotated, this would be the web page, but about the semantic structures in the background, i.e. the WSDL elements.

Thus, this step is about web service discovery by browsing and using information retrieval engines like Google as well as about reconciling semantic heterogenity between different web services, such as described in the WSDL documents and the web service ontologies they embed or refer to.

5.1 User Interaction

Service Browsing. With OntoMat-Service-Surfer the user can surf the service web, i.e. he can browse the web pages of web service advertisements and OntoMat-Service-Surfer highlights semantic annotations added by the web service provider. OntoMat-Service-Surfer indicates semantically-annotated web service elements, e.g. input parameters, by graphical icons on the web page. Thus, the user may easily identify relevant terminology that needs to be aligned with his own ontology.

As an alternative to deep annotation, the ontology browser in OntoMat-Service-Surfer may also visualize the underlying service ontology. OntoMat-Service-Surfer is able to interpret the description of web service operations and provide a corresponding form interface. The user may then directly proceed to web service invocation (Section 7) and invoke a concrete web service operations with data he provides via this generic form interface.

Deep Annotation. The user selects an ontology to be used for annotation and loads it into OntoMat-Service-Surfer. The user annotates the web service by drag'n'dropping highlighted items from the web page into the ontology browser of OntoMat-Service-Surfer. Doing so, he does not only extend the web page with metadata (which is possible), but most important here he establishes mappings between concepts, relations and attributes from the ontology used by the web service provider to his client ontology.

In the following we describe the deep-annotation of the vendor web service shown in Figure 9. The web page advertising the web service describes the `getLaptopOffer` operation and constitutes the context for the usage of the vendor ontology. The aim of the annotator is to translate the terminology used in the description of `getLaptopOffer` (cf. the WSDL document in Figure 2 and the vendor ontology in Figure 4) into his client ontology (Figure 6).

By drag'n'drop, one generates a graph of instances, relations between instances and attribute values of instances in the browser that visualizes the client ontology (cf. the left pane depicted in Figure 9).

When performing a drag'n'drop one will create a literal instance, if one drop's an instance of the vendor ontology onto a concept in the client ontology, or a literal value, if one drop's an attribute value of an instance onto an attribute in the client ontology. For instance, dropping 'IBM' onto the concept company would create a corresponding literal instance in the client ontology, dropping '7MB' creates a corresponding attribute value to a selected instance in the client ontology.

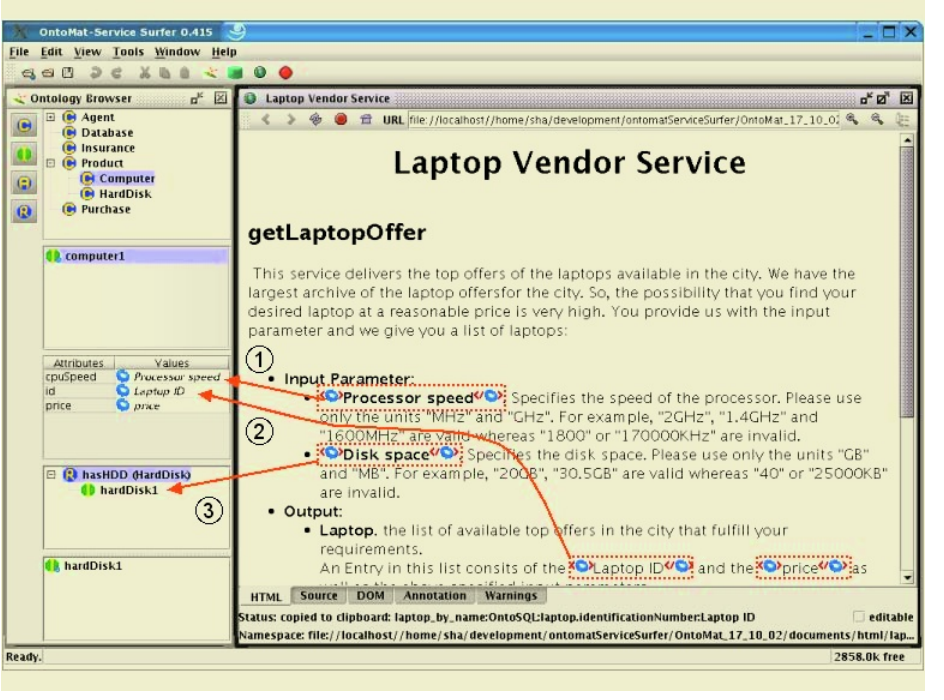

Fig. 9. Screenshot of OntoMat-Service-Surfer annotating vendor service

If one drop's a concept A from the vendor ontology onto a client ontology concept B, one will create a *generic instance*. A generic instance is just a variable that states that concept A in the vendor ontology corresponds to concept B in the client ontology.[3]

Thus, one may compile different types of instances in the client ontology. Each graph of non-separable, newly created instances and values in the client ontology corresponds to a mapping rule. For instance, one may *(i)*, drag'n'drop 'processorSpeed (from vendor ontology) onto cpuSpeed (from client ontology) that belongs to Computer (again in the client ontology). Thereby, *(ii)*, a generic instance is created for Computer with value Laptop (as cpuSpeed belongs to Computer and processorSpeed belongs to Laptop). The corresponding N3 notation reads as:

```
@prefix vendor: <http://laptop.wsdl/laptop/#>
@prefix client: <http://www.company.de/company.daml#>
vendor:Laptop a :GenericInstance; a client:Computer;
                                  client:cpuSpeed vendor:processorSpeed.
vendor:processorSpeed a :GenericInstance.
```

Its interpretation as a mapping is in first order logic:

```
FORALL X,Y (instanceOf(X,Computer) AND cpuSpeed(X,Y))
        <- (instanceOf(X,vendor:Laptop) AND vendor:processorSpeed(X,Y)).
```

[3] Corresponding generalizations exist for attributes and relationships.

One may trace the later drag'n'drop action in Figure 9, where action 1 picks up 'Processor Speed' with its underlying web service parameter processorSpeed (cf. the markup `elementURI="http://laptop.wsdl/laptop/#part(getLaptop OfferRequest/processorSpeed)"` in Figure 8). He drops onto the attribute that comes closest in his client ontology, viz. the aforementioned cpuSpeed, and generates the consequences just mentioned. Similarly, the second text item "Disk Space" being annotated with the input parameter `diskSpace` is handled in action 2. This time, however, the annotator must also create a hasHDD relationship between the generic instance hardisk1 and the generic instance of computer1 to build a larger graph representing a mapping rule with two generic attribute values (on cpuSpeed and diskSpace). Finally, the annotator maps the output parameters in action 3 (cf. Figure 9).

5.2 Mapping Rules Derived from Annotation

The results of deep annotation are mapping rules between the client ontology and each service ontology. The annotator may publish the client ontology and the mapping rules derived from annotations. This enables third parties (in particular logical roles that follow in the OntoMat-Service process) to execute the services on the basis of the semantics defined in the client ontology.

The mapping rules are defined in F-Logic. F-logic is a deductive, object-oriented database language that combines the declarative semantics and expressiveness of deductive database languages with the rich data modelling capabilities supported by object oriented model [8]. The annotator does not have to write F-logic rules. They are generated automatically by the OntoMat-Service-Surfer. Figure 10 and Figure 11 give the reader an intuition of how such automatically generated mapping rules look like when visualized with the OntoEdit plugins OntoMap (cf., [7]). Figure 10 shows the mapping from the company ontology to the vendor ontology which is a result from the annotation effort indicated in Figure 9. The result for the corresponding mapping of the insurer's ontology in depicted in Figure 11.

6 Web Services Planning

In this step, the web service end consumer selects the web service operations, he wants to use to accomplish certain tasks at hand. By making such a selection, he restricts the sets of relevant mapping rules. The inputs and outputs of web services are specified in the web service description documents of the web services. By considering the mapping rules and the information about the input and output types of web services, the planning component is able to infer valid web service flows.

If the output of a web service operation A is of type t and the input of another web service operation B is also of type t, then the service operations A and B can be plugged together (first A then B). Since, different web service providers will have different ontologies in general, this approach can only infer

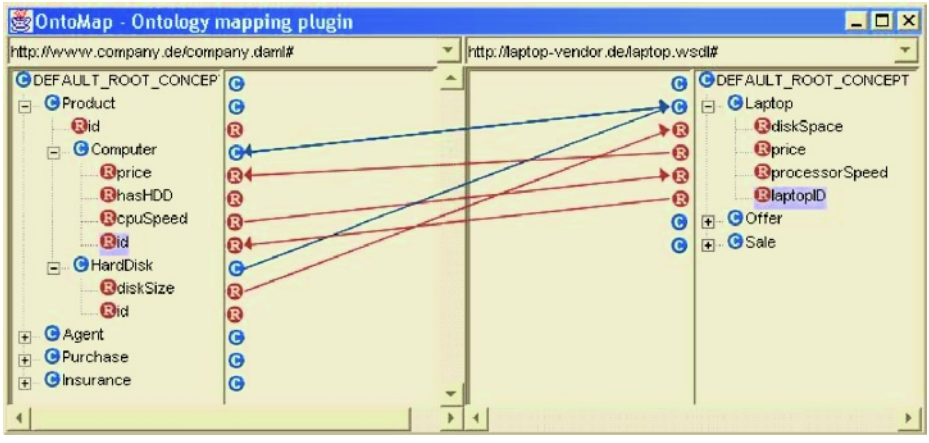

Fig. 10. Mapping between Client Ontology (left window) and Vendor Ontology (right window)

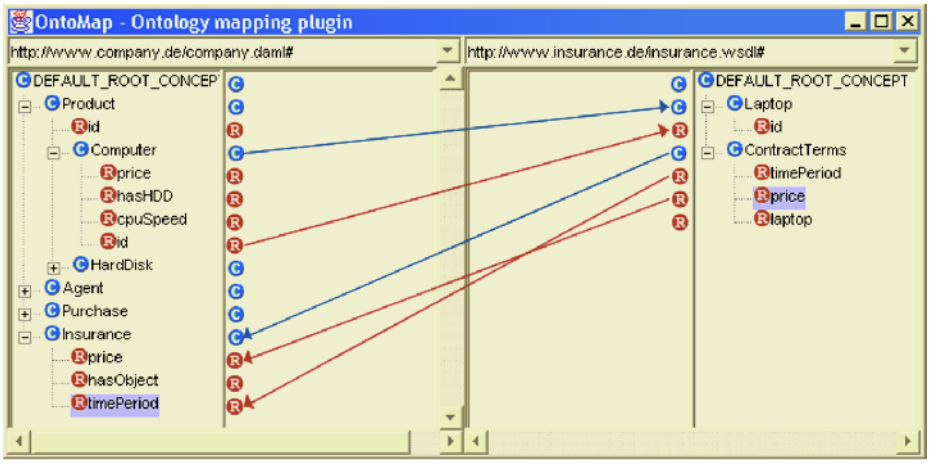

Fig. 11. Mapping between Client Ontology (left window) and Insurer's Ontology (right window)

web service flows in which all the web services are provided by one web service provider. However, web service flows consisting of web service operations from different providers can be inferred by using the restrictions contained in sets of mapping rules. For example, if the output of a service A is of type t_1 and the input of another web service B is of type t_2 and there is a mapping rule from t_1 to t_2, the services A and B can be plugged together (first A then B). In our running example, the service `getInsuranceTerms` can be called only after `getLaptopOffer`, since the former requires a `laptop`, which is the output of `getOffer`.

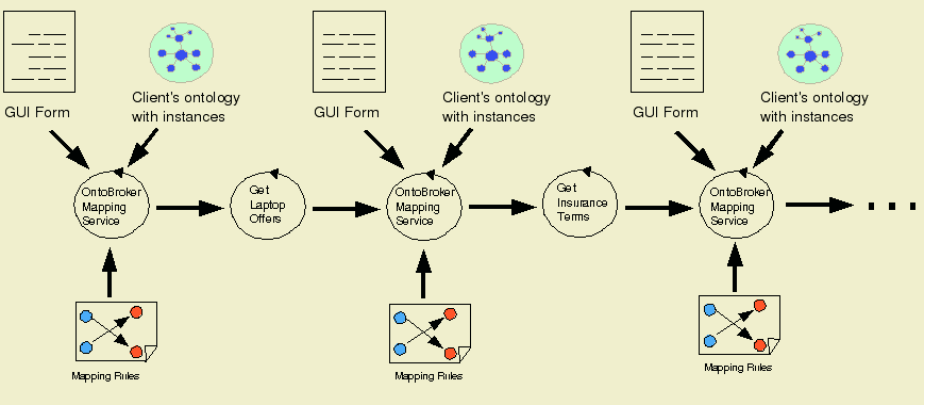

Fig. 12. Service Flow in our Running Example

7 Web Services Invocation

Finally, the user is presented with a list of feasible flows consisting only of web services selected by him in the previous step. The user chooses a flow from the list and invokes it.

The actual invocation is performed by a generic web services client engine. This engine takes a flow of web services as input. When the user requests the invocation of such a flow, the engine calls the web services in the order as specified in the flow. The invocation engine can be configured in a way such that data required from the client is retrieved automatically from client's ontology (with instances) during the invocation. The execution component communicates with OntoBroker [4], whenever mapping between concepts is required (cf. Figure 12).

The invocation component differentiates between the following cases (cf. Figure 12):

- **There are no mapping rules:** In this case, the user is provided with a form like interface, in which he has to enter required data according to the ontology of the respective web service provider to proceed the execution.
- **Automatic retrieval of data from client's ontology is not configured and mapping rules are defined:** In this case, the user is provided with a form like user interface, in which he has to enter required data according to his own (client's) ontology.
- **Automatic retrieval of data from client's ontology is configured and mapping rules are defined:** In this case, the invocation runs fully automatically.

This kind of approach is a generalization of common approaches to invocation of single web service operations. Let us consider this simple case in our framework: If a user wants to manually call only one web service operation, he will skip the definition of mapping rules. The flow will consist of only one web

service operation. When executing the single web service operation, the invocation engine will request data from the user via a form interface that reflects the ontology of the service provider (because no mapping rule exists).

8 Related Work

In this paper we provide an original framework, OntoMat-Service, to embed the process of web service discovery (here: by browsing web pages and retrieving web pages from search engines like Google), composition (here: by deep annotation and reasoning over logically possible configurations), and invocation (here: by OntoMat-Service-Surfer, and the mapping to a client ontology). The consideration of semantic heterogenity is germane to OntoMat-Service. It offers semantic translations as one of its core modules.

OntoMat-Service does not aim at a web service discovery, composition and invocation that is intelligent in the sense that it completely automates the task that typically the user is supposed to do. Rather, it provides an interface, OntoMat-Service-Surfer, that supports the intelligence of the user and guides him to add semantic information such that only few logically valid paths remain to be chosen from by the user.

To fully pursue such an objective, one needs a large set of different modules. We have built on our existing experience and tool framework for semantic annotation (cf. [6,7]) and for logical reasoning [4]. We have not yet dealt with the issue of web service flow execution and monitoring that is certainly needed to complement our current version of OntoMat-Service.

Closest to our approach come frameworks that facilitate the building of web service flows. A number of software systems are available to facilitate manual composition of programs, and more recently web services. Such programs, which include a diversity of workflow tools [18,5], and more recently service composition aids such as BizTalk Orchestration [10] enable a software engineer to manually specify a composition of programs to perform some task — though they typically neglect the aspect of semantic heterogenity that is core to OntoMat-Service.[4]

Web Services Invocation Framework (WSIF) [16] is an open source framework to execute any web service, that can be described by a WSDL document. However, it does not support the execution of a flow of web services.

Some technologies have been proposed that use some form of semantic markup of web services in order to automatically compose web services to perform some desired task (e.g., [13,3,12]). In [13], the authors use situation calculus for representing web service description and petri nets for describing the execution behavious of web services. In [3], the authors present an architecture of intelligent brokers that offer problem solving methods that can be configured and used by the users according to their needs. In [12] the authors propose an extended version of Golog for formalizing the provision of high-level generic procedures and customization of constraints. In [15], the authors propose a rule

[4] BizTalk even allows for XML-based (non-semantic) translations of data.

based expert system to automatically compose web services from existing web services.

On the one hand most recent experiences from such advanced projects like IBrow, however, have shown that these automatic composition techniques cannot yet been carried over to an open world setting. There one needs to tightly integrate the user of a web service — such as we do in OntoMat-Service. On the other hand OntoMat-Service can obviously be extended in the future to consider more types of automatic semantic matchmaking, service discovery [14,17] and configuration of web services into the web planning phase.

9 Discussion

In this paper we have described OntoMat-Service, an original framework to tie together the World Wide Web and web services into a Service Web. Germane to OntoMat-Service is its blending of browsing the Web, aggregating conceptual descriptions and web services and then investigating and invoking them from one platform.

We have also presented OntoMat-Service-Surfer, a tool that constitutes a prototype implementation of OntoMat-Service. Currently, our prototype understands WSDL with RDF(S) for web service descriptions, but its flexible architecture allows easy integration of more powerful web service description languages like DAML-S [2].

Clearly, one must be aware of what OntoMat-Service and OntoMat-Service-Surfer can do and what they can't do. OntoMat-Service is not intended to cater to businesses that want to establish durable, complex and high-quality web service connections with intricate interactions. For this objective, the integration by semantic annotation may provide a quick, first prototype, but semantic annotation cannot provide arbitrary complex mapping rules or arbitrary complex workflows. On the other hand, OntoMat-Service allows exactly for easily building a prototype web service integration and it allows for users with domain knowledge to participate in the Service Web — without programming.

OntoMat-Service opens up many interesting questions that need to be solved in the future, such as

- how to automate the way that Web Services are presented to the World;
- how to make compiled flows understandable to their users; or
- how to characterize the boundaries of what functionality can be aggregated and executed.

Eventually, OntoMat-Service and OntoMat-Service-Surfer, in conjunction with their counterparts in semantic annotation [6] and deep annotation [6], open up the possibility to bring Web pages, databases and Web Services into one coherent framework and thus progress the Semantic Web to a large Web of data and services.

Acknowledgements. A part of this work was funded by the SemIPort project of the federal ministry of education and research (BMBF).

References

1. Web service description language (wsdl) version 1.2, March 2003.
2. Anupriya Ankolekar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, Drew McDermott, David Martin, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, and et al. Terry Payne. Daml-s: Web service description for the semantic web. In *1st Int'l Semantic Web Conf. (ISWC 02)*, 2002.
3. V. Richard Benjamins, Enric Plaza, Enrico Motta, Dieter Fensel, Rudi Studer, Bob Wielinga, Guus Schreiber, and Zdenek Zdrahal. Ibrow3 – an intelligent brokering service for knowledge-component reuse on the world wide web. In *Proc.11th Banff Knowledge Acquisition for Knowledge-Based System Workshop (KAW98)*, 1998.
4. S. Decker, M. Erdmann, D. Fensel, and R. Studer. Ontobroker: Ontology based access to distributed and semi-structured information. In *DS-8 – Proceedings of the Conference on Database Semantics*, pages 351–369, 1999.
5. C.A. Ellis and G.J. Nutt. Modelling and enactment of workflow systems. *Application and Theory of Petri Nets*, LNCS 691:Modelling and enactment of workflow systems, 1993.
6. S. Handschuh and S. Staab. Authoring and annotation of web pages in cream. In *Proceedings of the 11th International World Wide Web Conference, WWW 2002, Honolulu, Hawaii, May 7–11, 2002*, pages 462–473. ACM Press, 2002.
7. S. Handschuh, S. Staab, and R. Volz. On deep annotation. In *Proceedings of the 12th International World Wide Web Conference, WWW 2003, Budapest, Hungary, May 20–24, 2003 (to appear)*. ACM Press, 2003.
8. Michael Kiefer, Georg Lausen, and James wu. Logical foundations of object oriented and frame-based languages. *Journal of the ACM*, 1995.
9. O. Lassila and R. Swick. Resource description framework (RDF) model and syntax specification. Technical report, W3C, 1999. W3C Recommendation. http://www.w3.org/TR/REC-rdf-syntax.
10. D. et al. Lowe. *BizTalk(TM) Server: The Complete Reference.*, November 2001.
11. Alexander Maedche and Steffen Staab. Services on the move – Towards p2p-enabled semantic web services. In *Proceedings of the 10th International Conference on Information Technology and Travel & Tourism, ENTER 2003, Helsinki, Finland, 29th–31st January 2003*. Springer, 2003.
12. S. McIlraith and T. Son. Adapting golog for composition of semantic web services. In *Proc 8th International Conference on Principles of Knowledge Representation and Reasoning*, 2002.
13. Srini Narayanan and Sheila McIlraith. Simulation, verification and automated composition of web services. In *WWW2002*, May 2002.
14. M. Paolucci, T. Kawmura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *First Int. Semantic Web Conf.*, 2002.
15. Shankar R. Ponnekanti and Armando Fox. Sword: A developer toolkit for building composite web services. In *Proceedings of the 11th International World Wide Web Conference, WWW 2002, Honolulu, Hawaii, May 7–11, 2002*. ACM Press, 2002.
16. Apache Web Services Project. Web services invocation framework.
17. Katia P. Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1):47–53, 1999.
18. van der Aalst and W..M..P. Woflan. A petri-net-based workflow analyzer, systems analysis – modelling – simulation. 35(3):345–357, 1999.