Discovering Web Services Using Behavioural Constraints and Ontology

Natenapa Sriharee and Twittie Senivongse

Department of Computer Engineering
Chulalongkorn University
Phyathai Road, Pathumwan, Bangkok 10330 Thailand
Tel. +66 2 2186991 Fax. +66 2 2186955
natenapa.s@student.chula.ac.th, twittie.s@chula.ac.th

Abstract. The ability to locate useful on-line Web Services is becoming critical for today's service-oriented business applications. A number of efforts have been put to enhance the service discovery process by using conceptualised knowledge, called ontology, of particular service domains to describe service characteristics. This paper presents an ontology-based approach to enhance descriptions of Web Services that are expressed in WSDL with ontology-based behavioural information, i.e. input, conditional/unconditional output, precondition, and conditional/unconditional effect of the services. Having a service ontology associated with each Web Service description, queries for services based on behavioural constraints can benefit from inferring semantics of the service from the service ontology. The service discovery process becomes closer to discovery by service semantics or behaviour, in contrast with discovery by matching of service attributes values – the mechanism that is supported currently by Web Services.

1 Introduction

Web Services are networked applications that are able to interact using standard application-to-application Web protocols over well-defined interfaces [1]. Standard Web Service architecture provides UDDI as a registry for service providers to advertise information about themselves and their services so that service consumers can search for the required providers and services [2]. The information model in UDDI roughly defines attributes that describe the service provider (i.e. business entity), the relationships with other providers (i.e. publisher assertion), the provided service (i.e. business service), and how to access the service instance (i.e. binding template). This set of information may refer to a tModel which is a specification of particular technical information (e.g. the interface and protocol used by the Web Service and expressed in WSDL [3]). Search can be done by name/key/category of business entities, services, or tModel. For example, a query could be "give me a list of providers who are in travel business" or "give me a flight booking service". UDDI will then match the attribute values as constrained in the query against those listed in the business or service descriptions.

J.-B. Stefani, I. Demeure, and D. Hagimont (Eds.): DAIS 2003, LNCS 2893, pp. 248–259, 2003.

The fixed set of attributes in UDDI limits the way queries can be composed. There are times when service consumers may look for a particular service with some semantics or behaviour. For example, a service consumer may want to find a flight booking service that can deliver the ticket to the consumer's address after payment has been made. It may be difficult for a service provider to describe such behavioural information in terms of attributes. Also, as UDDI places no requirement on a business service to be exposed as a Web Service, many service providers may use UDDI only as a channel to advertise their homepages. For this reason, WSDL which can be seen as describing behavioural information, although in a low-level form of interface signature and communication protocol, is not applicable in such a case, and therefore it is not used by UDDI when searching for services.

This paper adopts the idea of *Semantic Web* [4], which uses *ontology* to describe semantics of information, by supporting discovery of Web Services based on their ontological semantics. We define an upper ontology that models the capability of a Web Service. The capability is behaviour-oriented and represented by the service operation, input, conditional/unconditional output, precondition, and conditional/unconditional effect. Based on this upper ontology, a shared ontology of a particular service domain can be defined. Service providers can adhere to a shared service ontology to advertise the behaviour of their services. In the case that the shared ontology does not realise all detailed concepts of the service, the providers are allowed to extend it with a local ontology. We enhance the behavioural information of a Web Service expressed in WSDL by adding to it the ontology-based behaviour and present a framework for advertising and querying Web Services by behavioural constraints. Rule-based reasoning is also supported by the framework for determining the behaviour of the service when output or effect of the service are on some conditions. With this framework, behaviour-related query can be issued and service matching can take advantage of ontological inference.

Section 2 gives an overview of the ontology concept and Section 3 discusses related work on discovering Web Services using ontologies. Section 4 presents our upper ontology and service ontology and we show how ontology-based behavioural information can be added to a Web Service description in WSDL in Section 5. The framework for semantic service discovery is in Section 6 and we conclude the paper in Section 7.

2 Why Ontology?

An ontology is a formal explicit specification of a shared conceptualisation [5]. It was developed in Artificial Intelligence area to facilitate knowledge sharing and reuse. Fig. 1 shows an example of a NaturallyOccurringWaterSource ontology [6]. It shows common concepts or vocabularies (i.e. Class and Property) in such a knowledge domain and also the relationships between those concepts. A particular information instance or resource (i.e. Yangtze) can refer to its domain ontology and describe its own semantics. An inference engine can then infer more facts about the information instance (i.e. Yangtze is a Stream and a NaturallyOccuringWaterSource and EastChinaSea is a BodyOfWater).

```
(a) <?xml version="1.0"?>
    <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
       xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
       xml:base="http://www.geodesy.org/water/naturally-occurring">

     <daml:Class rdf:ID="NaturallyOccurringWaterSource"/>

     <daml:Class rdf:ID="Stream">
       <daml:subClassOf  rdf:resource="#NaturallyOccurringWaterSource"/>
     </daml:Class>

     <daml:Class rdf:ID="River">
       <daml:subClassOf rdf:resource="#Stream"/>
     </daml:Class>

     <daml:Property rdf:ID="emptiesInto">
        <rdfs:domain rdf:resource="#River"/>
        <rdfs:range rdf:resource="#BodyOfWater"/>
     </daml:Property>

     <daml:Property rdf:ID="length">
        <rdfs:domain rdf:resource="#River"/>
        <rdfs:range
              rdf:resource="http://www.w3.org/2000/01/rdf-schema#Literal"/>
     </daml:Property>
   ...
(b)  <River rdf:ID="Yangtze"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns="http://www.geodesy.org/water/naturally-occurring#">
       <length>6300 kilometers</length>
       <emptiesInto  rdf:resource="http://www.china.org/ geography#EastChinaSea"/>
      </River>
    </rdf:RDF>
```

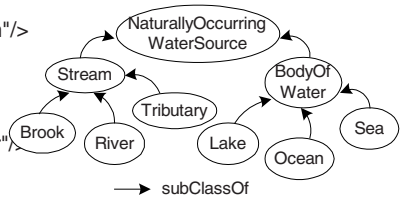

Fig. 1. NaturallyOccurringWaterSource in DAML+OIL (a) Ontology (b) Resource instance

Several XML-based markup languages are available for describing ontologies including RDF, RDFS, DAML+OIL, and OWL [4]. W3C has developed RDF with the goal to define a simple model for describing relationships between Web resources in terms of their properties and values. RDFS adds on to RDF the concept of classes of resources, class and property inheritance or subsumption, and domain and range of property. Based on top of these two languages, DAML+OIL [7] can additionally describe constraints on relationships of resources and their properties. These include cardinality, restrictions, and axioms describing, for example, disjunction, inverse, and transitivity rules. OWL is built upon DAML+OIL and will be a W3C recommendation for Web ontology language. To date, most of the available tools support DAML+OIL such as OilED and Protégé for ontology editing; Jess, JTP, BOR, and RACER for ontological reasoning; DQL and RDQL for querying. For the time being, we hence choose DAML+OIL for our knowledge representation.

3 Related Work

Using the ontology concept to enhance service discovery has now become a hot research topic. The work in [8] presents how a service registry can use an RDF-based ontology as a basis for advertising and querying for services. In [9], DAML project proposes a DAML+OIL-based language called DAML-S as a new service description language. DAML-S consists of the Service Profile ontology which describes functionalities that a Web Service provides, the Process ontology which describes services by a process model, and the Grounding ontology which describes transport details for access to services. The Service Profile ontology specifies the descriptions of the service and service provider, functional attributes such as service category and quality rating, and functional behaviour described in terms of the operation provided, input, output, precondition, and effect of the operation. Their consequent paper [10] shows a query to find a service with a required operation, input, and output, and a matching algorithm is devised based on ontological inference. The work in [11] annotates the operation, input, and output description of a Web Service, described in WSDL format, with DAML+OIL-based ontological concepts. Precondition and effect of the service are also added to WSDL as additional information, but they are not used for queries as only the matching of the operation, input, and output is considered.

Our work is very close to the work in [10] and [11]. Both consider behavioural aspects in their service models but those aspects are not fully considered or used as query constraints for service matching. In our work, we enhance WSDL with DAML+OIL-based ontological information and consider a Web Service by its behavioural aspects all round. We allow the operation, input, output, precondition, and effect to be used as query constraints, and additionally consider the case when output or effect of the service has some conditions placed on them - the case when we provide a rule-based reasoning to determine the output and effect for query matching.

4 Ontologies for Service Descriptions

Semantics of services can be described by *upper ontology* and *service ontology*. Upper ontology models general capability and behaviour of services while service ontology represents semantic concepts that are specific to application domains of services. Service ontology is further classified into *shared ontology* and *local ontology*. Shared ontology is common for service providers in a particular service domain whereas local ontology can be derived from shared ontology in order to represent further concepts of the service.

4.1 Upper Ontology

This upper ontology focuses on capability and behavioural aspects of a Web Service (Fig. 2). It refers to the classes of concepts that are related to a service including its service community, operations that it supports, data for input and output, effects of

the operations, and conditions before invoking the operation and for producing outputs or effects.

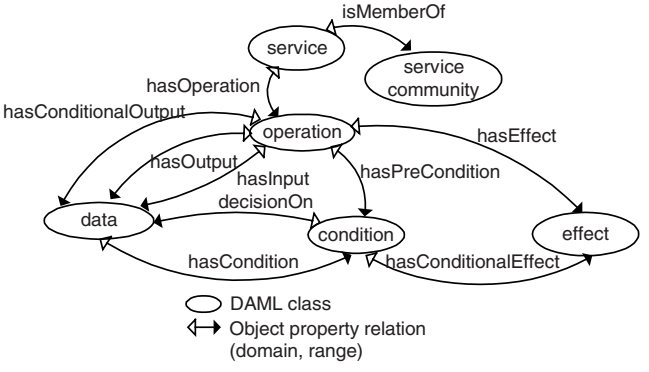

Fig. 2. Upper ontology for services

Properties of the concepts in the upper ontology consist of:

- *isMemberOf* specifies the service community to which the service belongs.
- *hasOperation* specifies an operation of the service.
- *hasInput* specifies an input data of an operation.
- *hasOutput* specifies an unconditional output data of an operation.
- *hasConditionalOutput* specifies a conditional output of an operation, i.e. the output that will be produced based on a certain condition.
- *hasPrecondition* specifies a condition that must be true before the execution of the operation.
- *hasEffect* specifies an effect or a change in the world after the execution of the operation.
- *HasConditionalEffect* specifies a conditional effect of an operation, i.e. the effect that will be produced based on a certain condition.
- *hasCondition* specifies a condition that must be true for producing a particular output or effect.
- *DecisionOn* specifies a resource or data whose value will determine the logical value of the condition for a particular output or effect.

4.2 Service Ontology

Semantic information specific to a particular service and common for service providers can be defined as a shared ontology. Service providers of the same service can refer to the shared ontology when creating their own WSDL descriptions. A shared ontology may be proposed by a group of service providers and built upon the upper ontology, some concepts that are shared with other ontologies, and the conceptual knowledge that is specific to this particular service domain. Fig. 3 (a) is an example of a shared ontology for a flight booking service.

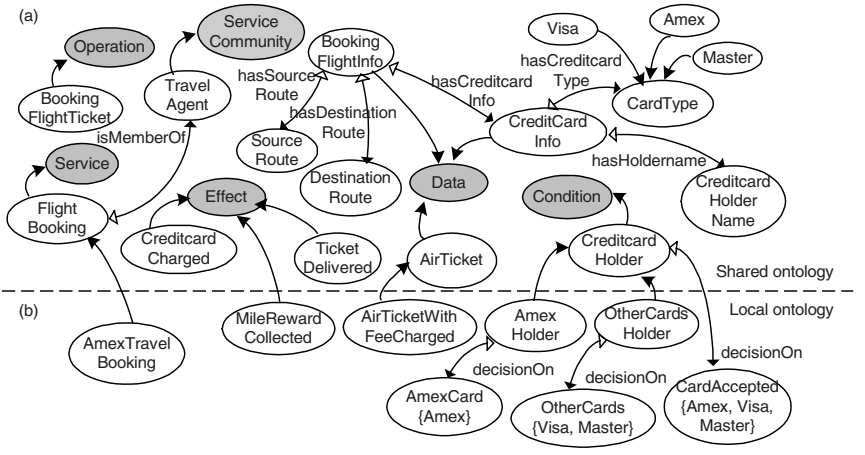

Fig. 3. Service contology. (a) Shared ontology for flight booking service (b) Local ontology for AmexTravel

Some of the semantic information in the shared flight booking ontology includes:

- *TravelAgent* that is a subclass of service community.
- *FlightBooking* that is a subclass of service and a member of *TravelAgent*.
- *AmexTravelBooking* that is a subclass of *FlightBooking* service.
- *BookingFlightTicket* that is a subclass of operation.
- *BookingFlightInfo* that is an input and a subclass of data, consisting of *CreditcardInfo*, *SourceRoute*, *DestinationRoute* etc.
- *CreditcardInfo* which is a subclass of data, consisting of *CardType* (such as amex, visa), name of holder etc.
- *AirTicket* that is an output and a subclass of data.
- *TicketDelivered* and *CreditcardCharged* that are subclasses of effect.
- *CreditcardHolder* that is a precondition and a subclass of condition, saying that the service consumer holds a credit card.
- *hasCreditcardType, hasHolderName, hasSourceRoute, hasCreditcardInfo etc.* which are properties of some classes.

Generally the basic idea is for all service providers to directly map their behaviour to one shared ontology of the service. However, it is possible that the semantics in the shared ontology may not contain all detailed aspects of the service behaviour and may not be mapped well for a particular service provider. In this case, the service provider may define a separate local ontology by deriving from the shared ontology of the service, and can use the concepts in both ontologies to declare links to the service behaviour. In Fig. 3 (b), a service provider called AmexTravel derives a local ontology from the shared flight booking ontology to refine some concepts. For example, *AirTicketWithFeeCharged* is derived from *AirTicket* to represent a special case of the output that the air ticket may come with extra fee. *MileRewardCollected* is another refinement, saying that by buying a ticket with AmexTravel, an additional effect is that the service consumer can collect mileage rewards. The condition *CreditcardHolder* is

refined by specifying the types of credit cards that AmexTravel can accept, i.e. *Amex*, *Visa*, or *Master*. *CreditcardHolder* is further derived into the conditions *AmexHolder* and *OtherCardsHolder* that will be used to determine different outputs of flight booking.

5 Adding Ontological Behaviour to WSDL

Concepts within the shared ontology and local ontology will be used to annotate WSDL descriptions of Web Services. We borrow the idea from [12] to add behavioural semantics to WSDL file and specify a mapping to link the service's own behaviour to the shared and local ontology. In this way, WSDL elements that represent the service capability (i.e. wsdl:message, wsdl:part, wsdl:operation, wsdl:input, wsdl:output) can be associated with semantic extensions that specify the ontological behaviour. Other semantics extensions (i.e. condition and effect) are added as extra elements. In Fig. 4, a shared ontology called flightbooking.daml is provided. A service provider named AmexTravel derives a local ontology called amextravel.daml. Its service provides an operation to book a flight ticket, the input is the required flight information, and the precondition is that the service requires a credit card for booking. The output is however conditional; if the consumer holds an Amex card, the output will be the air ticket; otherwise the output will be the air ticket and a service charge. The effects of the operation are that the ticket will be delivered to the consumer's address and the credit card will be charged. In the case that the consumer is an Amex cardholder, rewarding mileage points will be an additional effect.

6 Ontology-Based Service Discovery

The annotated WSDL descriptions will be used in a framework to discover Web Services. This section describes the discovery framework, how it integrates with UDDI, and an example of query with behavioural constraints.

6.1 Service Discovery Framework

The framework consists of several components that cooperate in semantic Web Service discovery. The prototype is Java Web-centric using Servlet technology and integrating existing ontology-supporting tools (Fig. 5). In step (a), an ontology engineer can use the *Ontology Builder* to build a shared ontology for a service. A service manager, in the same manner, can build a local ontology, based on the shared ontology, by using the Ontology Builder. OilEd [13] is adopted for Ontology Builder in our prototype although other ontology editors will do also. When ontologies are available, the service manager will, in step (b), add ontological information into an existing WSDL description by using *Semantic Mapper*. As a result, the Semantic Mapper will generate an annotated WSDL and an RDF service description which corresponds to the annotated WSDL. The service manager can also, in step (c), use the *Condition Builder* to translate preconditions and conditions for outputs and effects in the shared

ontology and local ontology into the rules for Jess engine [14]. Jess rules are stored in the *Rule DB* and will be used later to determine service behaviour at service matching time.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE uridef [
<!ENTITY sh "http://www.wssemantic.com/flightbooking.daml#">
<!ENTITY lo "http://www.AmexTravel.com/amextravel.daml#"> ]>
<definitions
   name="AmexTravel"
   targetNamespace="http://www.AmexTravel.com/amextravel.wsdl"
   xmlns:tns="http://www.AmexTravel.com/amextravel.wsdl"
   xmlns:WSDLext="http://www.wssemantic.com/WSDLext.dtd"
   xmlns:sh="&sh;"
   xmlns:lo="&lo;"  ...>
...
 <portType name="AmexTravelPortType">
  <operation name="BookingTicket"
    WSDLext:semantic-operation="&sh;BookingFlightTicket">
   <documentation>Provide service for booking air ticket
       </documentation>
   <input message="tns:BookingTicketRequest"
      WSDLext:semantic-data="&sh;BookingFlightInfo">
   </input>
   <WSDLext:precondition
       WSDLext:semantic-condition="&sh;CreditcardHolder"/>
   <output message="tns:BookingTicketResponse" />
   <WSDLext:conditionalOutput
       WSDLext:semantic-data="&sh;AirTicket">
    <WSDLext:condition
       WSDLext:semantic-condition="&lo;AmexHolder">
      <WSDLext:conditionalEffect
        WSDLext:semantic-effect="&lo;MileRewardCollected"/>
      </WSDLext:condition>
   </WSDLext:conditionalOutput>
   <WSDLext:conditionalOutput
       WSDLext:data="&lo;AirTicketWithFeeCharged">
    <WSDLext:condition
       WSDLext:semantic-condition="&lo;OtherCardsHolder"/>
   </WSDLext:conditionalOutput>
   <WSDLext:effect WSDLext:semantic-effect="&sh;TicketDelivered"/>
  <WSDLext:effect WSDLext:semantic-effect="&sh;CreditcardCharged"/>
   </operation>
  </portType>
   <binding name="AmexTravelBinding" type="tns:AmexTravelPortType">
    ...
  <service name="AmexTravel"
        WSDLext:semantic-service="&lo;AmexTravelBooking "/>
      <WSDLext:servicecommunity
        WSDLext:semantic-community="&sh;TravelAgent"/>
      <port binding="tns:AmexTravelBinding" name="AmexTravelPort">
       <soap:address
         location="http://www.AmexHolder.com/BookingAmexTicket"/>
      </port>
  </service>
</definitions>
```

Fig. 4. Extending WSDL of AmexTravel with ontological information

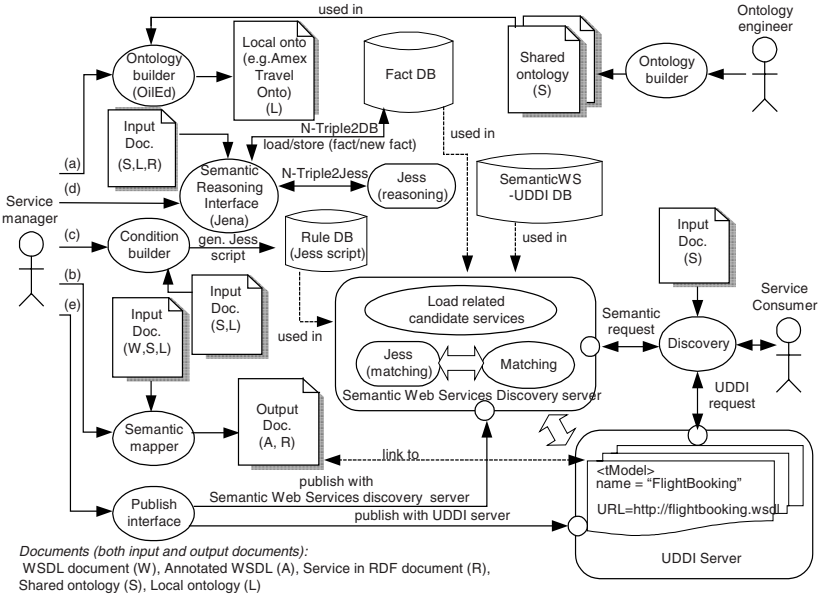

Fig. 5. Semantic Web Services discovery framework

In step (d), the RDF service description from step (b), the shared ontology, and the local ontology will be parsed by Jena module [15] within the *Semantic Reasoning Interface* in order to extract facts about the service. The facts are stored in the *Fact DB* in Jena's N-triple tuple format (i.e. <subject, predicate, object>). To reason for more facts, the Semantic Reasoning Interface is integrated with Jess which also provides a reasoning engine. The Semantic Reasoning Interface transforms existing N-triple tuple facts into Jess facts by using our predefined fact template, and then Jess engine can infer more facts especially those about the relations between the shared ontology and local ontology. Resulting facts from Jess will be transformed back to N-triple tuples and stored in the Fact DB. Facts in this DB will be consulted when matching a query against service descriptions.

In step (e), the service manager will publish service descriptions via the *Publish Interface*. A traditional Web Service description for a particular service provider (i.e. business entity, publisher assertion, business service, and binding template) will be registered with a UDDI server in a usual manner, with the annotated WSDL defined as a tModel for the binding template. To also provide for semantic discovery, an additional semantic entry will be registered with the *Semantic Web Services Discovery Server (SemanticWS-DS).* The semantic entry consists of the annotated WSDL and the reference keys to a particular business service and business entity in UDDI server. With these keys, after we query by service behaviour and a service provider is found a match, we can retrieve the complete information about this provider from the UDDI server. This semantic entry is stored in the *SemanticWS-UDDI DB*.

The SemanticWS-DS performs semantic service matching. A service consumer can compose an RDF query based on the upper ontology and shared ontology of the service. The SemanticWS-DS extracts information from the query and loads the facts about the services from the Fact DB to compare with the query. The resulting candidate services will be further checked by Jess rule engine to determine their exact behaviour against the constraints in the query. For those matched services, their complete description will then be retrieved from the UDDI server by using the reference keys stored in the SemanticWS-UDDI DB. The steps taken by the SemanticWS-DS are exemplified in Section 6.2.

6.2 Query by Behavioural Constraints

Suppose that a service consumer wants to query for a flight booking service that accepts a visa card, returns an air ticket as the output, and delivers the ticket to the consumer's address. The consumer will issue an RDF query based on the concepts in the upper ontology and the shared ontology of the flight booking service to the SemanticWS-DS. The information extracted from the query is in Fig. 6 (a). The consumer requests for a *FlightBooking* service in the *TravelAgent* community and the service has the operation *BookingFlightTicket*. The precondition *CreditcardHolder* and the *CardType* says that the operation must allow booking with credit card and visa card is accepted. The operation must return an *AirTicket* with an effect *TicketDelivered* for the consumer. This extracted information is queried against the facts about the available services that are stored in the Fact DB in N-triple tuple format. Consider some facts that are obtained by inferring from the shared flight booking ontology and the local ontology of AmexTravel in Fig. 6 (b). The service community, service, and operation concepts in the query match with those in the service description of AmexTravel. The output *AirTicketWithFeeCharged* of AmexTravel is defined as a subclass of the output *AirTicket* of the query so it is considered a match according to subsumption. Since *AmexHolder* and *OtherCardsHolder* defined in AmexTravel description are subclasses of the precondition *CreditcardHolder* of the query, AmexTravel is still a candidate service but further evaluation is required to check whether it accepts a visa card. The Fact DB returns all candidate services with exact or partial match with the query to the SemanticWS-DS.

The matching process continues by loading Jess rule scripts of the candidate services from the Rule DB in order to determine their behaviour that is based on some conditions. Fig. 6 (c) shows three Jess rules of AmexTravel. The information extracted from the RDF query is translated to Jess facts that are asserted into Jess rule engine. In this example, the assertion that specifies the precondition *CreditcardHolder* with the card type visa fires the *precondition-CardAccepted* rule, and therefore the precondition of AmexTravel satisfies the precondition of the query. Since this assertion also fires the *conditionalOutput-OtherCards* rule, all of its output and effects are compared with the output and effect specified in the query. The result is that the output *AirTicketWithFeeCharged* of AmexTravel matches the output *AirTicket* of the query by subsumption, and the effect *TicketDelivered* matches exactly. By satisfying all constraints, AmexTravel is returned as a search result to the consumer.

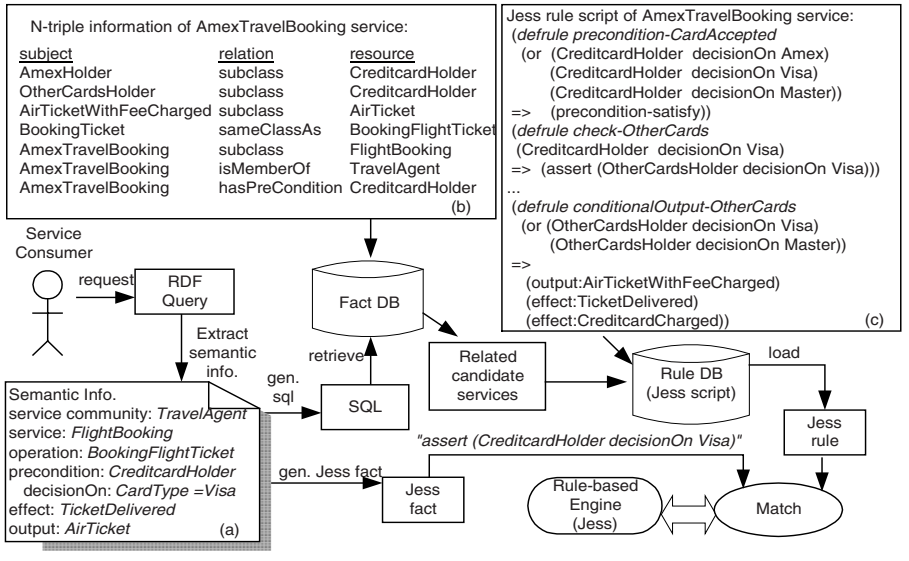

Fig. 6. Query by behavioural constraint (a) Semantics of query. (b) N-triple facts of Amex-Travel (c) Jess rule script of AmexTravel

7 Conclusion

This paper has proposed an approach to extend WSDL descriptions for Web Services with ontological information, based on an upper ontology, a shared service ontology, and a local ontology, in order to benefit from ontological reasoning. The approach allows queries for services based on capability and behavioural information such as the operation provided, input, output, and effect. Logical conditions can be placed on output and effect, and determined by a rule engine at query time.

Ranking query results by the degree of matching is an important issue that we will study further. Ranking may be based on the precedence assigned to those behavioural aspects in the upper ontology, relations between concepts, or the number of matched concepts and conditions. We also have a plan to explore the behavioural model that can enable automatic discovery of a group of services that altogether can satisfy a particular behaviour-related query. Such behavioural model is also expected to be useful for proving some behavioural properties of the service.

Acknowledgements

This work is supported by Thailand-Japan Technology Transfer Project and Chulalongkorn University-Industry Linkage Research Grant Year 2002.

References

1. W3C. Web Services (online). http://www.w3.org/2002/ws/
2. uddi.org. Universal Description, Discovery and Integration of Web Services (online). http://www.uddi.org
3. W3C. Web Services Description Language (WSDL) 1.1 (2001) (online). http://www.w3.org/TR/wsdl
4. semanticweb.org. Semantic Web (online). http://www.semanticweb.org
5. Gruber, T. R.: A Translation Approach to Portable Ontologies. Knowledge Acquisition Vol. 5 No. 2 (1993) 199-220
6. Costello, R., Jacobs, D.: RDF Schema Tutorial (online). http://www.xfront.com/rdf-schema/
7. daml.org. DAML+OIL (online). http://www.daml.org
8. Trastour, D., Bartolini, C., Gonzalez-Castillo, J.: A Semantic Web Approach to Service Description for Matchmaking of Services. Proceedings of the International Semantic Web Working Symposium (SWWS'01) (2001)
9. The DAML Services Coalition: DAML-S: Web Service Description for the Semantic Web. Proceedings of the 1st International Semantic Web Conference (ISWC 2002), Sardinia (Italy), Lecture Notes in Computer Science, Vol. 2342. Springer-Verlag (2002)
10. Paolucci, M. et al.: Semantic Matching of Web Services Capabilities. Proceedings of the 1st International Semantic Web Conference (ISWC 2002), Sardinia (Italy), Lecture Notes in Computer Science, Vol. 2342. Springer Verlag (2002)
11. Sivashanmugan, K. et al.: Adding Semantics to Web Services Standards. Proceedings of the International Conference on Web Services (2003)
12. Peer, J.: Bringing Together Semantic Web and Web Services. Proceedings of the 1st International Semantic Web Conference (ISWC 2002), Sardinia (Italy), Lecture Notes in Computer Science Vol. 2342. Springer Verlag (2002) 279-291
13. OilED, Version 3.5 (online). http://oiled.man.ac.uk/index.shtml
14. Jess the Rule Engine for the Java™ Platform, Version 6.1, 9 April 2003 (online). http://herzberg.ca.sandia.gov/jess/
15. Jena A Semantic Web Toolkit, Version 1.6.1 (online). http://www.hpl.hp.com/semweb/rdql.html