Improving Your Web Services Thorough Semantic Web Techniques

J.P. Liyanage, G.N. Wikramanayake
University of Colombo School of Computing, Colombo-7
Telephone: +94 0114 920562
Email: janakasoft@yahoo.com, gnw@ucsc.cmb.ac.lk

Abstract

Most of the data in the internet are markup for text and graphics in HTML format and are driven by syntax. These data are displayed to the user but the computers are concern there is no meaning of these data. If we semantically annotate data in the internet, then the computers can process these data in meaningful ways and increase the usability of the data. In this project a possible architecture for a semantics based middleware agent (broker) will be looked at, which has the capabilities to serve web service requests by searching through a larger space of web services. The matching capabilities of a semantics based broker are much higher than any of the syntax based middleware frameworks in the market today like UDDI.

In this project, we will be using two of the current semantic web technologies OWL (Ontology Web Language) and OWL-Services. Once the broker receives a client request that contains the capabilities and input/output specification of a web service client wishes to invoke, it will read OWL-S descriptions of the services registered to the broker and try to find a direct match. If it is unable to find one the broker uses its powerful inference capabilities (not available in conventional syntax based middleware) to dynamically compose a set of available services together into a new service.

Our IntelliBroker works as a true broker in the sense that client may even not know what the invoked services are. In executing the created composite service, the broker has the ability to exploit any parallelism between the child services and return results to the client faster. The broker will convert OWL types to SOAP messages and vice versa using XSLT (eXtensible Style Language Transformations). This paper focuses on the design and the architectural of our system.

We implemented the system with Java language and used Axis framework for publishing and executing web services. Jena API was used to manipulate OWL data and the Pellet engine was used for the reasoning.

Keywords: *Semantic Web, OWL-S, Semantic Web Services, Service Brokerage*

1. Introduction

Web services have been in the forefront of the web technologies in the recent past and allow much more rich interaction between the client and server, compared to the surfing of the HTML (Hyper Text Markup Language) pages scattered around the web. Since web services allows intelligent agents and/or applications in the web to access much more specific data compared to screen scraping methods used in conventional applications, it allows much more usable and targeted information to be forwarded to the end user. However, finding a suitable web service to satisfy a particular requirement had always been challenging. Registries containing information about many different types of web services are common in the World Wide Web. UDDI (Universal Description, Discovery, and Integration) is the most popular standard of describing such type of Web service description registries [1].

However all of these registries are syntax based, which mean that they are usually inflexible and inadequate. This also means that the data in these registries cannot be used to infer any other useful relationships between the web services than those explicitly defined in the registry. For example if the supplier A publishes a service to sell “Published Documents” and a user B wants to buy “Books” and do a search on “Books” he will not find the A’s web site. This is simply because there was no way to define the relationship between the “Published Documents” and “Books” in the syntax based Registries or more generally in the syntax based Web.

Semantic Web [11] attempts to solve these issues but is still in its infancy. The idea behind semantic web

has been to make machines understand the meanings (semantics) of the contents in the web and thus allowing them to infer any other implicit relationships between the data. Semantic Web proposes a language called OWL (Ontology Web Language) [18] to annotate semantically all the data published to the web.

Applying semantic web techniques to the conventional web services has resulted in what are known as semantic web services [10] and the aim of this project is to design a middleware agent (a broker) [20] for semantic web services. This brokerage system (IntelliBroker) process user requests for web services, and map them into one or more web services, which are capable of serving the request.

In addition to this, we have proposed a distributed architecture for a Broker. In this distributed architecture, rather than a single centralized broker handling all the requests, a set of loosely coupled brokers will share the knowledge between each other in serving the requests of clients. The brokers interact with each other using a semantics based protocol (called DUBIP) in the semantic web. Thus, the IntelliBroker is able to match more web services than a conventional broker based on syntax would.

The rest of the paper is organized as follows. A background on web services and semantic web is introduced along with related technologies. Then our methodology is explained. Our proposed distributed architecture is described next. The implementation of our ideas and the results are then presented. Finally conclusions and future work is highlighted.

2. Background

2.1. Web Services

Web services are built mainly upon XML and HTTP [20]. HTTP (Hyper Text Transfer Protocol) is a ubiquitous protocol, running everywhere on the Internet while XML (eXtensible Markup Language) provides a meta-language using which you can write specialized languages to express complex interactions between clients and services or between components of a composite service. Using XML, other technologies like WSDL [2] and SOAP [20] are built, which further defines the platform elements of web services.

Simple Object Access Protocol (SOAP) is a protocol specification that defines a uniform way of passing XML-encoded data. It also defines a way to perform Remote Procedure Calls (RPCs) using HTTP as the underlying communication protocol [20].

WSDL

Web Service Description Language (WSDL) [2] provides a common XML grammar for describing services and a platform for automatically integrating those services; thus it is a mandatory component for any web service. Using WSDL, a client can locate a web service and invoke any of its publicly available functions [2]. Since machines can understand and process WSDL specification, it is possible to automate the process of integrating with a new service (e.g. generation of a proxy class in Microsoft .NET [13]).

RDF

Resource Description Framework (RDF) [12] is used as the foundation for representing metadata about Web resources such as the title, author, and modification date of a Web page, copyright and licensing information about a Web document [12]. It provides syntax for expressing simple statements about resources, where each statement consists of a subject, a predicate, and an object (triples). In RDF resources are identified by URIs (Universal Resource Indicator) and thus can be used to describe web based resources effectively.

RDFS

One limitation of RDF is that it does not have any facilities for defining structures or hierarchies [8]. RDFS (RDF Schema) was thus introduced as an extension to RDF, to complement RDF with a type system. It provides the facilities needed to specify classes and properties (defined as a directed binary relation) in RDF. In RDFS, a set of new terms has been introduced to define classes, subclasses and properties applicable to them.

2.2. Semantic Web

First introduced by Tim Berners-Lee [11], it is about a new form of web content that would be meaningful to computers and thus allow computers to infer meaningful relationships between data. Semantic Web technologies are supposed to be a solution to the problem of allowing the data in the internet available to a much broader range of consumers (either human or machines) preferably through automated agents. Semantic web is an extension to the current Web, where information is given a well-defined meaning and thus machines can process and “understand” the data rather than merely displaying them like in the current Web [11].

The Semantic Web principles are implemented in layers of Web technologies and standards [Figure 1]. The Unicode and URI layers make sure the use of international characters sets and provide the method of

identifying objects in Semantic Web. The XML layer with namespace and schema definitions assures the integration of Semantic Web definitions with the other XML based standards. With RDF and RDFS, it is possible to make statements about objects with URIs and define vocabularies that can be referred to by URIs. This is the layer where we can give types to resources and links (properties). The Ontology layer supports the evolution of vocabularies as it can define relations between the different concepts. With the Digital Signature layer, agents or end users can detect any alterations to documents.

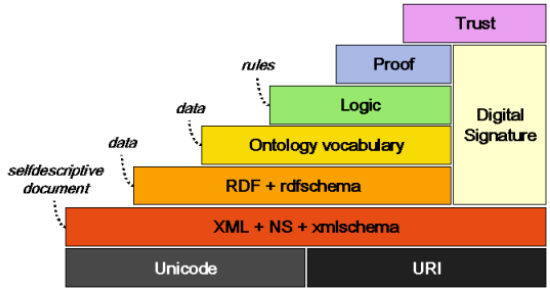

Figure 1: Layers in the semantic web

The top three layers (Logic, Proof and Trust), are currently being researched under W3C and simple application demonstrations are being constructed. The Logic layer enables the writing of rules while the Proof layer executes these rules. Trust layer provides a mechanism to determine whether to trust a given proof or not [10]. Proof and Trust are very important concepts in Semantic Web since if one person says that X is blue and another says that X is not blue, we need a way to determine which is true [14].

2.3. OWL

The Semantic Web needs a support of *ontologies*, which is defined as explicit specification of a conceptualization [5]. Ontologies define the concepts and relationships used to describe and represent an area of knowledge. OWL has been designed and introduced by W3C to be able to describe ontologies. OWL (Ontology Web Language) is a layer on top of RDFS, which defines an additional set of terms to describe the relationships between the resources in a much richer fashion.

Sub languages

To compromise between rich semantics for meaningful applications (expressive power) and feasibility/implementability the OWL support three different sublanguages Lite, DL and Full [6, 18].

Classes and Properties

OWL extends the notion of classes defined in RDFS, with its own construct. OWL also extends the notion RDFS property by introducing Data-type and Object properties. OWL also introduces a set of property characteristics, which results in considerable inference capabilities [7]. For example, Colombo region can be located in the Asia region if given that Sri Lanka is located in Asia region and Colombo located in the region Sri Lanka.

2.4. RDQL

Resource Description Query Language (RDQL) is the most commonly used query language to obtain information from databases containing RDF data like OWL or OWL Services. The work of RDQL is similar to the work of SQL in relation to the relational databases [17].

2.5. OWL-S

OWL-S (OWL Services) is a set of ontologies designed in OWL to describe web services in semantic web. By using this common set of ontologies, the web services will be universally available to any client that can understand OWL. Actually there is another framework called WSMO (Web Service Modelling Ontology), aiming to describe semantic web services [15]. However since OWL-S is the most widely accepted choice for semantic web services and is the recommended one by W3C, we used OWL-S for service description in this project.

Service Profiles, Models, and Groundings

The top level of the OWL-S ontology is the Service class, which corresponds to a web service. One needs to know at least three things about a Service: what it does, how it works and how one might access it. The relationships between these classes are depicted in Figure 2.

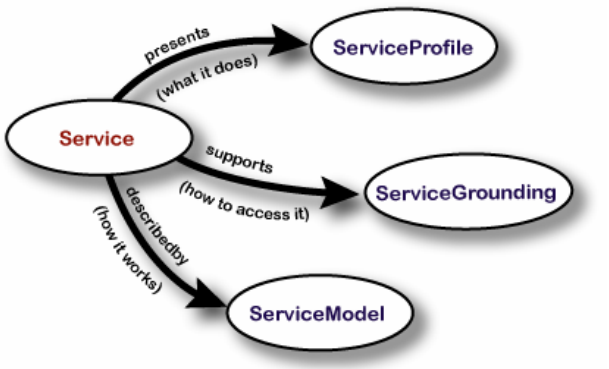

Figure 2: Top-level class relationships in OWL-S

2.6. Web Service Brokerage

A web service broker finds suitable web services for the client based on the capabilities requested by the client. The tasks performed by a broker are twofold:

Search: Finding the best service(s) matching to the requesters query. The broker may have to compose existing services together in order to derive new services, which will satisfy the query.

Mediation: Converting back and forth between the data formats used by requester and provider.

Matching capability of two brokers can be compared using the number of matches given for a particular query. If the number of matches given is higher for Broker A than for Broker B then broker A is said to be better than B. This number depends both on the search and mediation capabilities of the broker. That is the amount of the services searched and the ability to mitigate the differences between the consumer and provider.

In addition to these main functionalities, a broker should be secure (in the sense only authorized parties can access it). A broker can also provide services like web service promotion, negotiation and maintaining QoS (Quality of Service) metrics of registered services.

2.7. Current Brokerage Systems

Complete brokerage systems that are capable of doing all of the tasks of discovery (matchmaking), service composition, mediation and execution automatically without human intervention are not available to date. There are partial solutions, which implement parts of the above functionalities with the human intervention developed using OWL-S. We will explain three of the most popular systems, namely: XPlan, CMU OWL-Broker and OWLS-MX.

XPlan

This tool is a composition-planning engine for semantic web services, which are described in OWL-S. It converts OWL-S 1.1 services to equivalent problem and domain descriptions that are specified in the planning domain description language (PDDL), and invokes an AI planner (XPlan) to generate a service composition plan sequence that satisfies a given goal [9]. However, this tool is intended only for expert users and provides a nice GUI to manipulate the composed services manually. It only allows compositions with sequences of child services. Additionally the tool has the ability to consider QoS metrics in selecting suitable candidate services for the composition [4].

CMU OWL-Broker

This is a prototype system developed by the W3C designers of the OWL-S specification[16]. They have based their implementation on a generic OWL-S processor called OWL-S VM. The broker's activities are divided into two parts namely the advertisement protocol and mediation protocol. In the advertisement protocol the service providers' service metadata are collected in internal broker registries. In the mediation protocol following activities is performed:

- The requester query the broker using OWL Query Language (OWL-QL)
- The broker searches for a matching provider using the advertisements in its registry
- After selecting the best provider, the query is mapped into the inputs expected by that provider.
- Upon receiving the reply from provider, it is mapped back into the outputs expected by requester.
- Finally, the output is sent back to the requester.

This broker does not have the ability to create (by composing) new web services and matchmaking is limited to the services registered to the broker. It also does not take the QoS metrics into consideration when finding the matches.

OWLS-MX

This is a hybrid semantic web service discovery tool in the sense it uses both semantic reasoning and syntactic based similarity metrics to obtain the best of both worlds [19]. The service descriptions must be in the OWL-S format and the system is implemented in Java. It uses the OWL-DL reasoner called "Pellet" for semantic reasoning. This is a very recent system developed in 2006. However, this has only the discovery capability and human intervention is required throughout the process of selecting services to testing whether the matched services are correct. However, the program has a very easy to use GUI, which allows us to visualize the whole process of matchmaking [3].

3. Methodology

In order to derive a model to implement the required functionalities, we design a system called DUBIP that has three main components:

- The client agent or service requester
- The web service broker
- The server or service provider

The high-level interactions among these three components are given in the Figure 3 below.

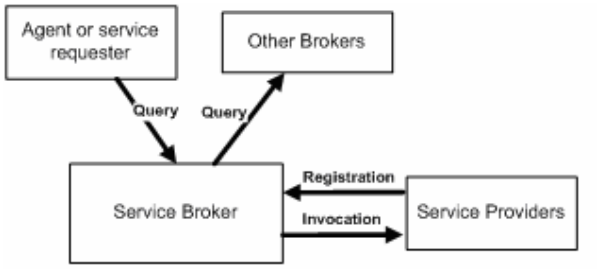

Figure 3: High-level components and Interactions

Here, first the client send a query stating the requirements for the service he wants to invoke and the data required to invoke the service. The broker will search through the registered services and invoke a service that has capabilities to satisfy the client request. Service providers register the services, by submitting the descriptions of their available services. In addition, DUBIP introduces other brokers into the system, so that those will be referred if a matching service is not found in the current broker.

We have used the OWL-S specification version 1.1 to describe the provider services to the broker. These semantic descriptions made available to the broker, will make it possible to find matching services to the requester query and to compose new services to satisfy the query. In order to find matches, the broker needs to do reasoning, which requires that the broker can access all the ontologies referred by the service descriptions. An example plan generated by the broker to satisfy a client query of booking a cheapest flight from Colombo to New York is given in Figure 4. Here four web services has been used sequentially and iteratively to create a new service.

The agent at the client side needs to know the semantic markup in order to formulate a query to be sent to the broker and to process the results sent back by the broker.

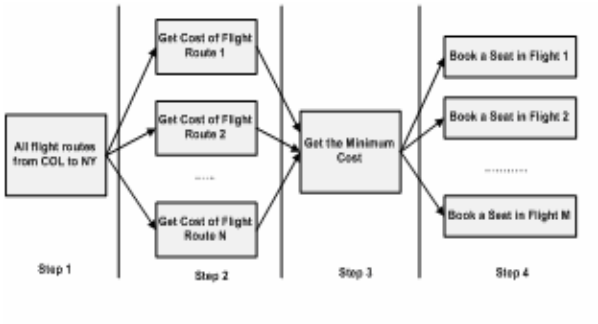

Figure 4: A composition plan for a query

4. Architecture

In the following sections, we have given a brief description of the architecture of the three main components of this brokerage system.

4.1. Client (requester)

The agent's query should contain two different types of information:

- A description of the capabilities of the service

This is related to the metadata of the service we are looking for. It will be a set of OWL classes, which will describe the preconditions and effects of the service we are searching.

- Input data for the invocation of the service

This is a set of OWL class instances, which are supplied as a possible set of inputs for the service, the client is expecting to execute.

Here the broker will return an error code to the client if a service with the requested capabilities are not found. It is up to the client agent to handle these errors in their own ways.

To handle complex message handling with the broker, a client-side component is designed. This component will simplify the usage of the broker by the user applications, which does not need to know about semantic markup or existence of multiple brokers. With respect to the DUBIP this client-side component is required to handle the following tasks:

- Selecting the nearest or fastest broker
- Selecting the next available broker if the current one fails

The main processes of the client-side component are depicted in the Figure 5. The client-side component should be a platform dependent system, which needs to interact with the user applications very closely.

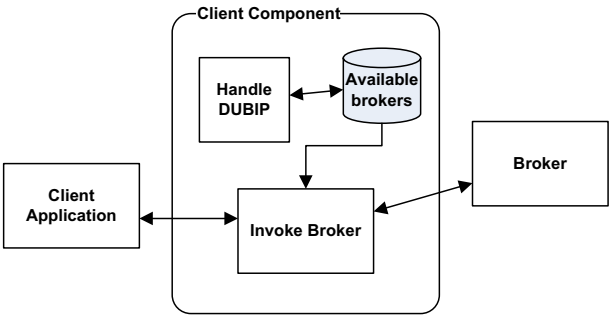

Figure 5: Processes in the Client-side component

4.2. The Broker

This will be the core component of the semantic brokerage system, which will handle the main tasks of registration, discovery, composition and mediation. While handling these tasks the broker must be aware of the other brokers and co-exist with them. To implement the complex set of requirements to be fulfilled by the broker, it is divided into a set of tightly integrated set of modules. These modules and the interactions among them are depicted in Figure 6.

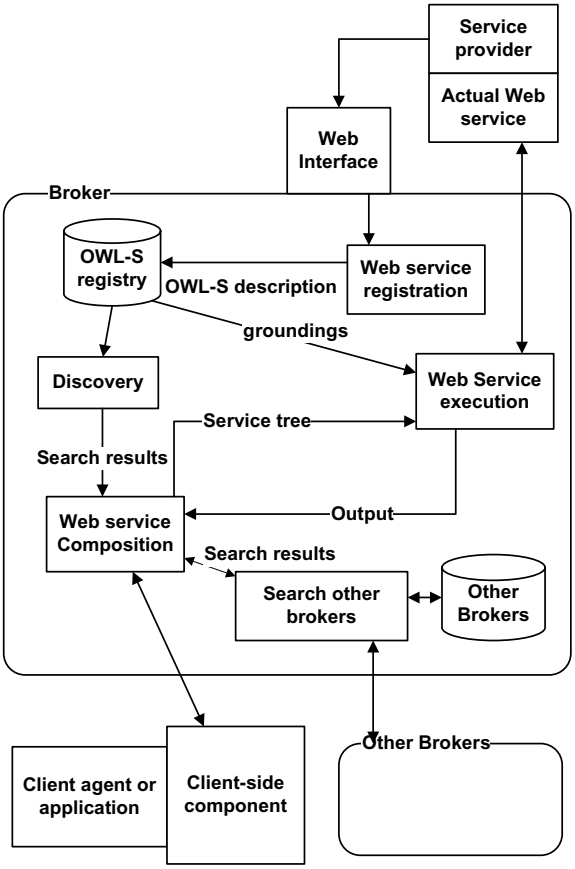

Figure 6: Main modules inside the brokerage engine

Service Registration

New web services will be registered by submitting the OWL-S description of the service with a service provider. A provider can register more than one service and before the service is registered, the provider is registered with the broker.

The OWL-S description of a service may refer to custom ontologies designed by the provider. In this case, the referred ontologies can either be made available through an URL or submitted as a part of the registration request. The OWL-S description and the custom ontologies will be stored as files in the file system of the broker.

Providers can register either Atomic services, which can be executed directly or Composite services, which can only be executed after matching atomic parts are found. An Atomic service is registered by submitting an OWL-S document containing a Service, which is `DescribedBy` a single `AtomicProcess`. Similarly, a Composite service is registered by submitting an OWL-S document containing a Service that is `DescribedBy` a single `CompositeProcess`.

Upon registration the broker will need to update QoS information for each of the invocation on the service. That is if the service execution was not successful for some request then the Quality metric of that particular service will be reduced. If the Quality metric of the service is reduced below a specified threshold, the service will be unregistered from the broker and the relevant service provider will be notified using the contact details in the database.

Service Composition

The query from the requester is received by this component and will be handled mainly by this component before passing onto other components. Upon receiving the query from the requester, the actions taken by this Composition component are depicted in Figure 7.

The Service Composition component works as the mediator between the activities of all other components in the Broker, as can be seen from the Flow chart in Figure 7. The processes 1, 2 and 5 of Figure 7 are executed by the "Service Discovery" component. The processes 4 and 6 are parts of the "Service Execution" component while the process 3 is a part of the "Search other Brokers" component.

An important thing to notice here is that we always prefer Atomic services to Composite services in searching for matches. The composite processes mentioned here are of two types:

- The skeleton services registered by the providers
- Services dynamically generated by the composition engine

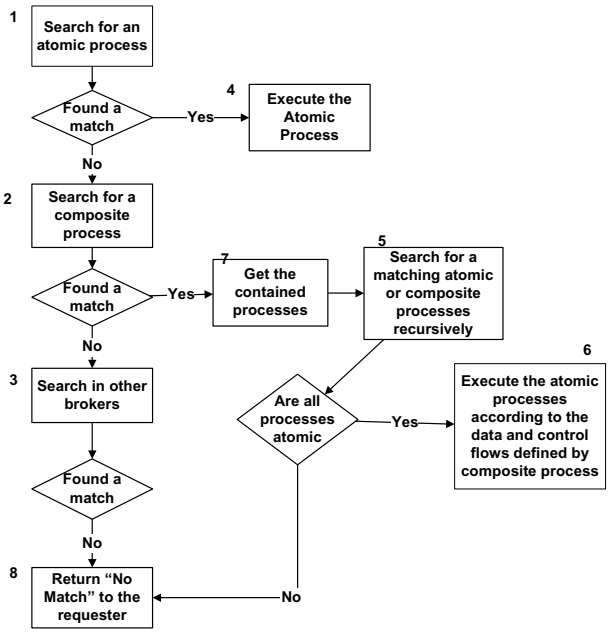

Figure 7: Flow chart showing control flow among the components of broker

For the Composition component to return a successful response to the requester it should have found at least one Atomic service and zero or more Composite services, which specifies the data and control flows among these Atomic services.

One important activity handled by this Composition component which is not shown in Figure 7, is the asking for more inputs from the requester in case the inputs provided by requester are not sufficient to invoke the selected service. The requester will be asked for more inputs only if the requester has capabilities to provide inputs on demand (i.e., if the mode-2 is used as described previously).

Service Discovery

The service discovery would be done by checking the `ServiceProfile` of the OWL-S documents stored in the OWL-S registry. The preconditions and effects (indicated by `hasPrecondition` and `hasResult` respectively) of the `ServiceProfile` will be match with those specified in the query to find a match or a set of matches. In addition, if more than one composite service is found, the services with least number of parts are preferred and thus searched for matching Atomic parts firstly.

The Atomic and Composite services discovered to satisfy a query, can be thought of as making up a tree hierarchy in which Atomic services make up leaves with Composite services making up non-leaves.

Figure 8 shows a possible hierarchy of services discovered to match the query described in Figure 4. The boxes in thick borders denote Atomic services while boxes with thin borders denote Composite services. Here the service A is found by the process 2 in Figure 7 and then the other services B-G are found by the process 6 in Figure 7 by recursively searching for matching parts in all the composite services. After all the Composite services are decomposed into Atomic services the root service (e.g. service A in this case) which satisfies the query, can be executed since the WSDL groundings are available to all the lower level Atomic services.

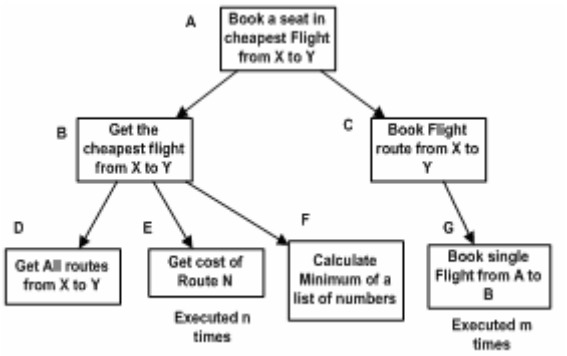

Figure 8: Tree of services discovered to satisfy a Query

In matching inputs, outputs, preconditions and effects by comparing the OWL classes of these instances, there are three possible options namely exact, relaxed, and subsume. In exact matching required and available class are equal. In relaxed matching required type is a sub class of the available type and vice versa in subsume matching.

Service Execution

After the Tree of services satisfying the query has been found by the "Service Discovery" component, it is forwarded to this component. Here for each of the Atomic services, the groundings will be obtained from the OWL-S registry by reading the `ServiceGrounding` from each of the specific OWL-S documents. The `ServiceGrounding` specifies the URL of WSDL document and the XSL transformations (which defines the conversion between OWL data types and WSDL data types). The WSDL document specifies the location of the concrete web services and thus the web service can be executed.

The executions of the services in Tree hierarchy will proceed in Bottom-Up order, feeding the Output of the lower level services to the upper level service, until the Output of the root level service is available. This will be returned to the requester as the response to the query.

The execution component exploits parallelism among the services to reduce the service time. If two services are not dependent on each other (through output to input forwarding), the execution engine

identifies that sends SOAP requests to both the services at the same time.

Search other Brokers

This module searches for matching services in other brokers known to it. This is how the overall process happens:

- A limited number of brokers (n number of brokers) known to the broker will be stored in the "Other Brokers" registry shown in Figure 6.
- The list of brokers is refreshed frequently to store the n closest brokers to this broker.
- Upon receiving the search request from Composition component, the search request is sent to the closest broker (broker B) to this broker (broker A).
- Broker B searches its registry for a matching service. If one is found it would be sent to A. Otherwise B will send the search request to its closest broker exempting A (say C). The process continues until a match is found or a threshold number of brokers are searched.
- Say at broker M a match is found (or threshold), then M will directly send the reply to the original broker A (or no match error if threshold).

5. Implementation

We have implemented the system using Java as our main programming language. We have used the Jena API to manipulate OWL descriptions associated with services are well as to reason about the service descriptions and other custom ontologies.

The Java web services were implemented using the Axis framework and hosted in Apache Tomcat server. We used the same Axis framework to execute provider services. The broker does matching based on inputs, outputs and effects of web services. Matching based on preconditions or complex SWRL expressions are not implemented. Dynamic composition of web services by the broker was limited to sequences of services. However, providers can design abstract composite services using any of the control structures available in the OWL-S specification and register them with the broker. The broker will match the slots in the composite service with concrete atomic services.

We also developed a prototype client component to send requests to the broker and display the responses. Figure 9 shows this client GUI application. It has five main tabs, three tabs aiding the user to formulate the request and two tabs to display the response sent back by the broker. "Inputs" tab allows the user to define input types and actual values associated with these types, while "Outputs" and "Effects" tabs let the user define output and effect types respectively. The user can also specify the matching strength used in comparing the input and output types for matches. "Equivalent" denotes the strictest match condition where the types should be the same for a match.

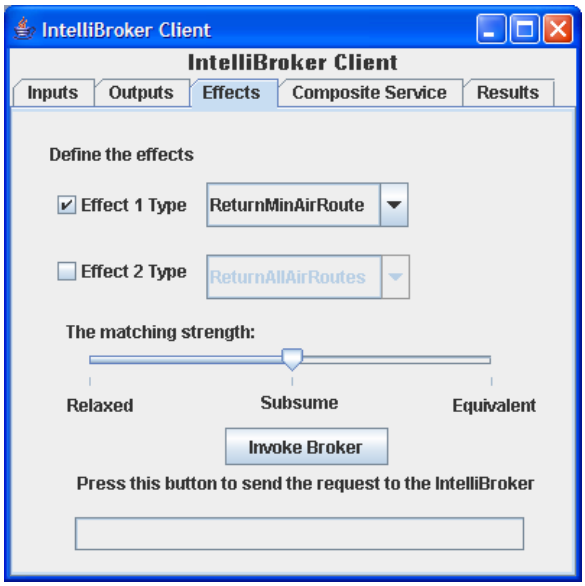

Figure 9: The client GUI for formulating requests to be sent to the broker

After invoking the broker, the resulting composite service created by the broker is shown in the fourth tab and the actual response, which contain the output data values are shown in the fifth tab.

6. Results

We tested the brokerage system by using number of web services from a flight-booking scenario. We invoked the broker using various inputs, outputs and effects using the prototype client component we have implemented. We have also used super classes and sub classes of the input & output types with relaxed matching strength to test the mediation capabilities of the broker.

The system worked reliably and predicatively for all the services in our sample scenario. Figure 10 shows a composite service generated and subsequently executed by the broker. However, the sample scenario contained only half a dozen of services and about a dozen of data types. The importance of using effects to describe services will increase as the number of registered services rises since more IO based matches would result in a large search space.

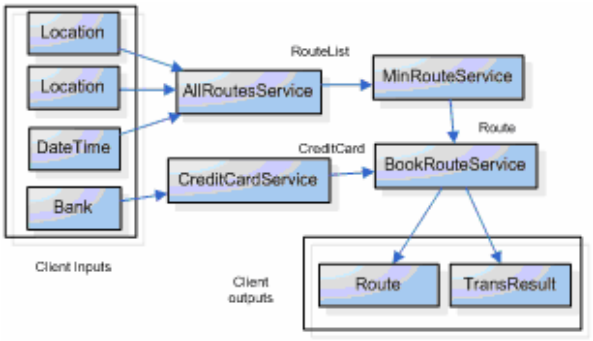

Figure 10: A composite service generated dynamically by the broker

7. Conclusion

This project was concerned with using semantic web related technologies to dynamically select and compose web services to satisfy client requests and finally execute these web services to return the result to the client. Even if semantic web is out of the research labs, semantic web services are not. We used the W3C recommended semantic web services language called OWL-S to semantically annotate our web services. OWL-S specification is still under the R&D stages and going through rigorous changes as new features are added and existing features are altered. Even if OWL-S specification as it currently stands is very rich in describing the web services, reliable third party library support for manipulating these semantic descriptions are not available as of yet. Another consideration is that using these rich descriptions in general contexts such as public web is still not very practical as many web-based agents are still not capable to reading these specifications. The broker we have implemented in this project is concerned with some core features of the OWL-S specification and is suitable for a specific context or a domain, where the available core OWL-S features can be used in an intuitive manner to obtain predictable results (a mandatory requirement for any enterprise scale application). In that sense, this project works as an ambitious effort to bring the newest developments in the academia through to the industry.

Another important fact is that when someone in the industry wants to do something related to semantics (for example in Enterprise Application Integration) they always go for their own proprietary/custom formats rather than going with available standard formats, due to the complexity and generality of these formats and the performance considerations of implementing these full standards. We think this project keeps an important step towards identifying a useable set of core features to be used in industry oriented, semantic web service related applications.

7.1. Future Work

The importance of OWL-S to semantic web services is analogues to the importance of WSDL to the syntactic (conventional) web services. The ability of the current WS frameworks to automatically generate WSDL for their web services is a key factor for the popularity of the current web services. Humans intervention will be needed only choose the OWL types for WSDL complex types and to define preconditions and effects. All of these can be done through an easy to use GUI without requiring the user to ever look at the WSDL or OWL-S files directly.

OWL-S precondition and effects (called post-conditions as well) are currently based on SWRL expressions and hard to be understood by conventional procedural programmers (let along expecting them to write such expressions). However even the W3C activity group is still not sure of what language to be used to define preconditions and effects and it is a still wide open research area to select a suitable candidate. However, as soon as the research community and the W3C come to a conclusion of what rule language to be used for OWL-S preconditions and effects, and as tool support for manipulating rule expressions is available, this functionality can be integrated to the IntelliBroker.

Due to the time constraints, we did not implement the multiple broker interaction functionality. However, in a distributed environment like internet (where failures are common) it makes sense to have number of small service brokers interacting with each other rather than having one single broker creating a single point of failure. Since the DUBIP itself is based on semantic web languages, rather than syntactic fluff (like HTTP or SOAP) it is inherently extendable and this introduces us to the new concept of semantic protocols.

Acknowledgement

We want to thank all of the people who have contributed with solutions to the OWL-S mailing list. We want to thank Harshana Liyanage at Zone24x7 Inc for providing us with insights into semantic web. Finally yet importantly, we wish to thank all who have contributed to the progress and improvement of semantic web technologies, which has inspired us carry on this project from start to end.

References

1. ARIBA. 2000. UDDI Overview. Fujitsu Limited.
2. CERAMI, E. 2002. WSDL Essentials. *Web Services Essentials 6*. O'Reilly & Associates.

3. FRIES, B., AND KLUSCH, M. 2006. OWLS-MX, Hybrid Semantic Web Service Matchmaker, Version 1.1b. Users Manual.

4. GERBER, A., BUTT, S., AND KLUSCH, M. 2006. OWL-S XPlan, OWL-S Semantic Web Service Composition Planner. DFKI Saarbrücken, Germany.

5. GUNATILAKA, M., WIKRAMANAYAKE, G. N., AND KARUNARATNA, D.D. 2004. Implementation of Ontology Based Business Registries to Support e-Commerce. University of Colombo School of Computing. Proceedings of 6th International Information Technology Conference, Colombo, Sri Lanka, 29th Nov – 1st Dec 2004, pp. 222-231.

6. HERMAN, I. 2005. Tutorial on Semantic Web Technologies. WWW Consortium.

7. JENA 2.2 DOCUMENTATION. 2005. RDQL Examples, HP Research Labs.

8. JIAN, S., SUI, X., AND WANG, B. 2004. Semantic Web Services.

9. KLUSCH, M., GERBER, A., AND SCHMIDT, M. 2005. Semantic Web Service Composition Planning with OWLS-XPlan. Proceedings of the 1st Intl. AAAI Fall Symposium on Agents and the Semantic Web, Arlington VA, USA, AAAI Press, 2005

10. KOIVUNEN, M., AND MILLER, E. 2001. W3C Semantic Web Activity. In *Proceedings of Semantic Web Kick-off Seminar 2001.* WWW Consortium.

11. LEE, T. B., HENDLER, J., AND LASSILA, O. 2001. Semantic Web. *Scientific American (May)*.

12. MANOLA, F., AND MILLER, E. 2004. RDF Primer**.** W3C Recommendation 10 February 2004. World Wide Web Consortium.

13. MSDN. 2003. .NET Web Services. *MSDN Documentation*. Microsoft Cooperation.

14. PALMER, S. B. 2001. The Semantic Web: An Introduction. Internet. Retrieved from http://infomesh.net/2001/swintro on 2nd September 2005.

15. PAOLUCCI, M., AND SYCARA, K. 2002. DAML-S: Semantic Markup for Web Services. The DAML Services Coalition.

16. PAOLUCCI, M., SOUDRY, J., SRINIVASAN, N., AND SYCARA, K. 2004. A Broker for OWL-S Web services. The Robotics Institute, Carnegie Mellon University.

17. ROMAN, D., ZAREMBA, M. 2005. Semantic Web services for Autonomic Computing. Second IEEE International Conference on Autonomic Computing, Seattle, USA. June 2005.

18. SMITH, M. K., WELTY, C., AND MCGUINNESS, D. 2004. OWL Web Ontology Language Guide. W3C Recommendation 10 February 2004.

19. SYCARA, K., FRIES, B., AND KLUSCH, M. 2006. Automated Semantic Web Service Discovery with OWLS-MX. Proceedings of 5th International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS), Hakodate, Japan, ACM Press

20. VASUDEVAN, V. 2001. A Web Services Primer. Internet. Retrieved from *http://webservices.xml.com/pub/a/ws/2001/04/04/webservices on 01st September 2005*.