

Accuracy in Selecting Reconfigurable Web Services

M N Sahulamid¹ and M Regina Bagam²

¹ SBM college of engineering and technology

Received: 6 December 2012 Accepted: 31 December 2012 Published: 15 January 2013

Abstract

Service-Oriented Architecture (SOA) provides a flexible framework for service composition. Using standard-based protocols (such as SOAP and WSDL). There are several constraints meant for selecting the right and appropriate service to be designed as reconfigurable dynamic web services. Those constraints leverage to the following factors availability, response time, failure handling and supports dynamic configuration. Our paper presents the way of predicting the service methods which are really necessary for providing as a dynamic web service. Since all the service methods cannot be used as dynamically as it depends upon the number of users really using the service by the service providers.

Index terms—

1 I.

2 Introduction for Selecting

Dynamic Web Services eb Services are software applications or services that are uniquely identified by a URI (Uniform Resource Identifiers) and expose public interfaces for clients, using XML (extended markup language). Those web services can be discovered and used by other client applications using XML based messages and protocols such as HTTP.

The emergence and continued development of web services standards such as SOAP (simple object access protocol) and WSDL (web services description language) [3] enable us to request and describe web services in a standard way. This will increase the ease of use of web services, enable interoperability between heterogeneous platforms and help businesses solve Authors ? ? : SBM college of engineering and technology, India. E-mail : sahulamid@gmail.com integration problems of their applications. Consequently, it is anticipated that web servers that host the services will be subject to increasing usage and have a higher load. Furthermore, the current simple modulus operand involving client/server activation of a single web service will be enhanced to support more complex scenarios, in which applications and service providers them selves rely on other external web services as part of their business logic. The reliance on third party web services reduces the control of the Organization over its application and (sometimes) mission-critical code. The control and information of certain parts of the system is pushed outside organizational boundaries. Scenarios involving reliance on external web services raise several new issues and challenges. An example of common scenario would be of clients consuming external web services, which in turn outsource their computational resources to other service providers. Furthermore, runtime information such as service load and availability or business related constraints might affect the selection process of an external web service, and not be predecided, as it is today. In the existing frameworks for web services there is no incentive to bind dynamically to a specific web service. However, once runtime information concerning those web services is available to the application, a dynamic binding becomes advantageous over a static, pre-decided one. We suggest a model that provides the web service client runtime information that is pertinent to its execution and business logic. The client application can then dynamically bind to the temporarily best service, from a selection of acceptable web services it works with, and according to the client's set of constraints. A client may want to apply some business rules when dynamically choosing a web service, or may be more concerned with response time or availability. When response time is critical (e.g. stock quotes service etc.) it is important

for an application to activate the fastest web service available at that given time, or have some mechanism that ensures availability and reliability. When several clients participate in such a scenario, an indirect load balancing mechanism is created, which helps to direct clients to available and relatively fast web services.

Figures 1 illustrate a client activation decisions based on information gathered at runtime from the service providers according to the client constraints. In figure 1, the client is concerned with availability and response times of a web service; after retrieving related information from the service providers, it activates the fastest available web service. This behaviour contributes to the robustness of the client application. Figure 1 shows client activation, based on response time and quality of service. According to the client's business constraints, it may prefer to switch to another service provider when it observe a change in the combination of quality and response time offered by the service providers. The recommendation component asks the user to rate the executed service, so it will be used for recommendation purpose.

3 II.

4 Related Work

Architecture

5 Semantic Matcher

Service providers publish DAML-S [5] descriptions of services to a Semantic Description Repository. A service user gives his requirements using DAML-S description. The semantic matcher finds the match between user requirement and all published service descriptions using a Semantic Matching Algorithm. It along with Recommendation System gives matching services in an order.

Figure ?? shows the detailed architecture of a Semantic Matcher [6][7] [4]. The Ontology Inference Engine creates a knowledge base from the specified ontology in a DAML-S description and a request description. Web Service Description parser parses the Web Service Descriptions to find out different parameters to be matched. The criteria table specifies service attributes to be compared and the If the first conceptual annotation contains the second, the mapping is called a Container map and if first conceptual annotation is part of the second, the mapping is called Part of map. Otherwise the mapping is called disjoint map.

6 Recommendation System

The Dynamic Web Service Selection Framework has are commendation system, which recommends the best service satisfying the user's requirements. When a user uses a web service, it asks user to rate a web service; so that users can help each other to find a better web service. This is especially important when there are more than one web services which have same functionality but their quality of service is different. We provide the user, a metric to help him decide the rating of a web service. It will be a comparison matrix of runtime behavior of a web service and the users expected QoS parameters like max execution time, average execution time, max response time, average response time etc. Web service with better quality of service will get more rating than other service which offers same functionality but poor service quality. The recommendation system uses the item based collaborative filtering approach [8]. As users rate web services, it is possible to predict how a given user will rate a particular web service. Once it knows prediction of ratings to each web service satisfying user requirements, it can recommend web services in order of their ratings. This approach looks at the set of web services the target user has rated and computes how similar they are to the web service for which user rating is to be predicted. Once the similar web services are found, the prediction is computed by taking a weighted average of the target user's ratings on these similar web services. The item based collaborative filtering approach has two aspects namely similarity computation and prediction generation.

7 a) Similarity Computation

The similarity [9] between two web services is computed by subtracting the average rating of the two web services. Considering only users who have rated both web service A and web service B, say that there are 10 such users, we sum the ratings that A and B got, say 65 and 85. Clearly B is ranked higher than A by 2 on average. The similarity between web services is computed whenever users rate a web service. The result of similarity computation is stored in a similarity matrix.

8 b) Prediction Generation

The prediction function [8][9] predicts how a particular user will rate a web service. It computes prediction on a web service i for a user u by computing the sum of ratings given by the user on the web services similar to i . Each rating is weighted by the corresponding similarity $S_{i,j}$ between web services i and j . $P_u, i = \frac{\sum_{j \in \text{similar items}} (S_{i,j} * R_{u,j})}{\sum_{j \in \text{similar items}} S_{i,j}}$ Basically it tries to capture how the active user rates the similar web services. The weighted sum is scaled by the sum of the similarity terms to make sure the prediction is within the predefined range. If the user has used a similar service, it predicts his likely satisfaction index for this service/service chain. If no similar service has been used before, it considers the average rating of all the users for similar services.

9 III.

10 Dynamic Web Service Invocation -Advanced a) Headers

Besides parameters, a web service operation may include "headers". Headers are basically additional parameters that are carried inside the header of a SOAP request/response instead of in the body. In general headers are used to specify additional information not strictly related to the semantics of an operation such as the credentials (username and password) required to invoke it.

The WSDData class allows managing parameters and headers homogeneously: while the voidset Parameter(<parameter-name>, <parameter-value>) AbsObjectgetParameter(<parameter-name>) String get Parameter String(<parameter-name>), Intget Parameter Integer(<parameter-name>), booleanget Parameter Boolean(<parameter-name>) methods are available to manage parameters, the voidset Header(<header-name>, <header-value>) AbsObjectgetHeader(<header-name>) String getHeaderString(<header-name>) integergetHeaderInteger(<header-name>) booleangetHeaderBoolean(<header-name>) methods are available to manage headers.

11 b) Proxy

In many cases both the access to a WSDL (at Dynamic Client initialization time) and the actual web service invocation require passing through an HTTP Proxy. The Dynamic Client class provides the following methods to set proxy information.

? setProxyHost(<host>): Set the proxy host (e.g. 163.162.10.12) ? setProxyPort(<port>): Set the proxy port (e.g. 8080)

? setNonProxyHosts(<listOfAddresses>): Set a list of addresses (possibly including '*') that will be accessed without using the proxy. The separator is the '|' character ? setProxyAuthentication(<user>, <password>): Set the credentials (if any) required to access the proxy The following code snippet provides an example. dc.setProxyHost("10.12.175.14"); dc.setProxyPort("8080"); dc.setNonProxyHosts("163.163.*|.telecomitalia.it"); dc.setProxyAuthentication("myUser", "myPwd"); dc.initClient(new URI("http://myWSDL")); c) Security Certain web services require HTTP Basic Authentication. The Dynamic Client class provides the following methods to set HTTP related information.

? setDefaultHttpUsername(): Specifies the http username used in all requests.

? setDefaultHttpPassword(): Specifies the http password used in all requests. The following code snippet provides an example. dc.setDefaultHttpUsername("MyHttpUsername"); dc.setDefaultHttpPassword("MyHttpPassword");

If the credential of HTTP Basic Authentication are different in all requests is possible specify them in invoke(?) method with Security Properties object.

Instead, if the credential of HTTP Basic Authentication are different for the WSDL discovery is possible specify them in initClient(?) method.

The following code snippet provides an example dc.initClient (new URI("http://myWSDL"), "MyHttpUsername", "MyHttpPassword"); Other web services require WS-Security Username Token. The DynamicClient class provides the following methods to set WSS related information.

? setDefaultWSSUsername(): Specifies the wss username used in all requests. ? setDefaultWSSPassword(): Specifies the wss password used in all requests. ? setDefaultWSSPasswordType(): Specifies the wss password type used in all requests (TEXT or DIGEST, see SecurityProperties object). The following code snippet provides an example. dc.setDefaultWSSUsername("MyWSSUsername"); dc.setDefaultWSSPassword("MyWSSPassword"); dc.setDefaultWSSPasswordType(SecurityProperties.PW_TEXT);

If the credential of WS-Security Username Token are different in all requests is possible specify them in invoke(?) method with Security Properties object.

Other web services require WS-Security Timestamp. The Dynamic Client class provides the following method to set WSS related information.

? setDefaultWSSTimeToLive(): Specifies the wss request time to live (in second) used in all requests. The following code snippet provides an example. dc.setDefaultWSSTimeToLive(60);

If the credential of WS-Security Timestamp are different in all requests is possible specify them in invoke(?) method with Security Properties object.

Other web services require SSL connections with or without certificates. The Dynamic Client class provides the following methods to set SSL related information.

? enableCertificateChecking(): Enables the certificates checking mechanism. When this mechanism is ? setTrustStore(<file.keystore>):

Specifies the keystore holding certificates of trusted remote servers ? setTrustStorePassword(<password>): Specifies the password used to protect the keystore of trusted certificates The following code snippet provides an example. dc.setTrustStore("C:/myFolder/cert.keystore"); dc.setTrustStorePassword("myPassword"); dc.initClient(new URI("http://myWSDL"));

12 d) Caching

Considering that the initialization of a Dynamic Client (`initClient()` method) is a long operation that may take some seconds, a good approach is to create a single Dynamic Client instance for each WSDL and reuse it whenever an operation of a service described in that WSDL must be invoked (note that the `invoke()` methods of Dynamic Client class are thread safe and therefore can be called by two or more threads in parallel). In order to facilitate this practice the WSDC provides a class called Dynamic Client Cache that manages all issues related to creation, initialization and caching of Dynamic Client objects in a thread safe mode. The Dynamic Client Cache class follows the singleton pattern and therefore the first step when using it is to retrieve the singleton Dynamic Client Cache instance by means of the `getInstance()` method.

The following code snippet shows how to use the `DynamicClientCache` class.

```
DynamicClientCache dcc = DynamicClientCache.getInstance();  
DynamicClient client = dcc.get(new URI("http://myWSDL")); WSDData output = client.invoke("sum",  
input);
```

The `get()` method of the `DynamicClientCache` class first checks if a `DynamicClient` object was already created to access the given WSDL and returns it in that case. Only if no `DynamicClient` object is already available a new one is created and initialized.

13 IV.

14 Service Selection Algorithms for General Flow Structure

Many real-world service processes have services that are not in strictly sequential order. They may have parallel operations to perform several services at the same time, conditional branch operations, and loops for using a service more than once in a flow. The function graph for composite service with general composition patterns may contain complex structures among function nodes. In order to simplify the problem and construct a service candidate graph with a DAG structure, we first remove the loop operations by unfolding the cycles as in [Zeng et al. 2004]. A cycle is unfolded by cloning the function nodes involved in the cycle as many times as the maximal loop count.

V.

15 Conclusion

We have studied the problem of service selection with multiple QoS constraints and proposed several algorithms. The selection of dynamic web service is depends upon the Execution price, Execution duration, Reputation, Successful execution rate, Availability, response time ? 600, cost ? 25 0, availability ? 85%.



Figure 1: Figure 1 :

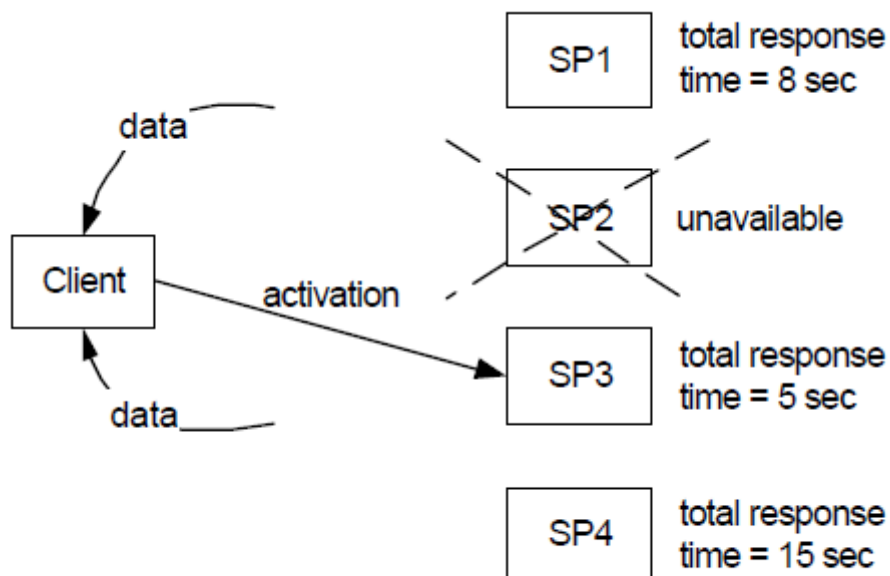


Figure 2: Figure 2 :

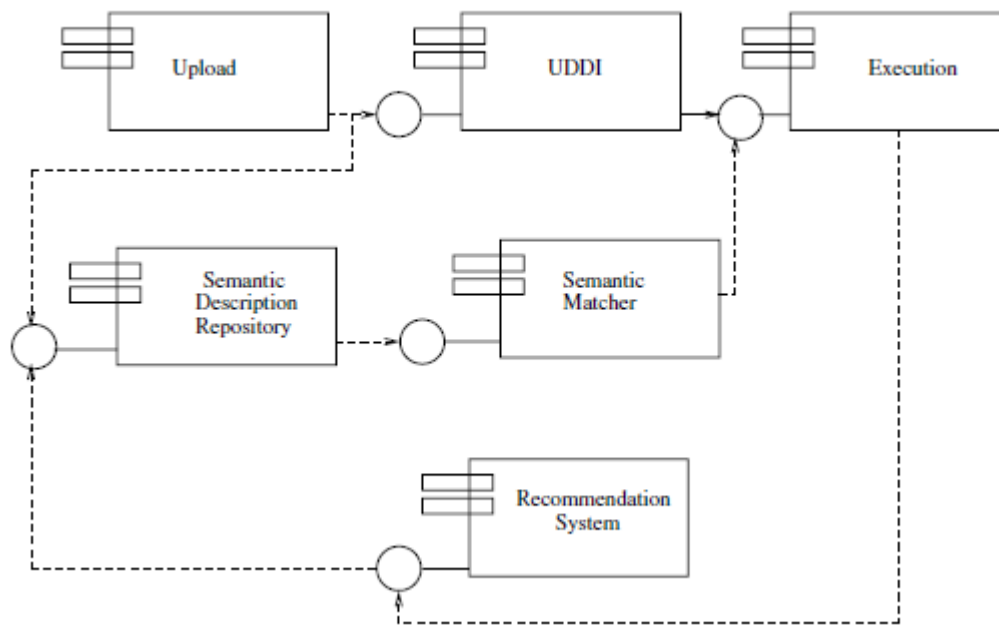


Figure 3: E

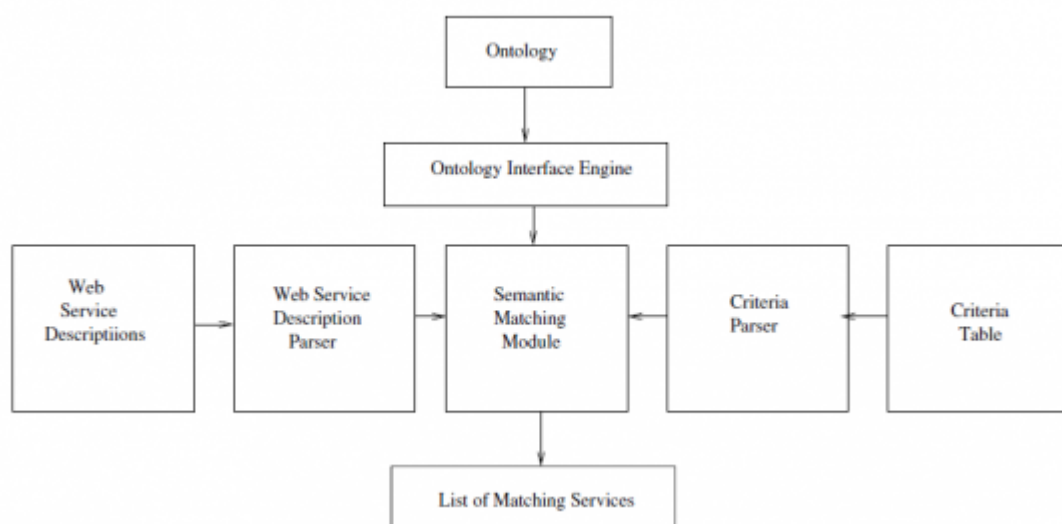


Figure 4: E

189 [DAML Technical Committee. DARPA Agent Markup Language-DAML] , <http://www.daml.org> *DAML*
190 *Technical Committee. DARPA Agent Markup Language-DAML*

191 [Tao et al.] *Efficient Algorithms for Web Services Selection with End-to-End QoS Constraints*, Y U Tao , Yue
192 Zhang , Kwei-Jay Lin . University of California, Irvine

193 [Evrensirin and Hendler ()] ‘Filtering and Selecting Semantic Web Services with Interactive Composition Tech-
194 niques’. Bijanparsia Evrensirin , James Hendler . *IEEE Intelligent Systems* 2004. 19 (4) p. .

195 [Lemire and Mcgrath (2005)] *Implementing a Rating-Based Item-to-Item Recommender System in PHP/SQL*,
196 Daniel Lemire , Sean Mcgrath . D-01. January, 2005. (Technical Report)

197 [Sarwar et al.] *Item-based Collaborative Filtering Recommendation Algorithms*, Badrul Sarwar , George Karypis
198 , Joseph Konstan , John Riedl . (In the Proceedings)

199 [Kanpur shripad@cse.iitk.ac.in T.V. Prabhakar Indian Institute of Technology] *Kanpur shripad@cse.iitk.ac.in*
200 *T.V. Prabhakar Indian Institute of Technology, Kanpurtvp@cse.iitk.ac.in* Manikrao Indian Institute
201 of Technology

202 [Doshi et al. (2004)] *Parameterized Semantic Matching for Workflow Composition*, Prashant Doshi , Richard
203 Goodwin , Rama Akkiraju . March, 2004. (IBM Research Report, RC23133 (W0403-026))

204 [Paolucci ()] ‘Semantic Matching of Web Services Capabilities’. M Paolucci . *The Semantic Web-ISWC 2003: 1st*
205 *Int’l Semantic Web Conf*, 2002. Springer-Verlag. 2342 p. 333.

206 [Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) Available Markup Language (XML) ()]
207 ‘Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL) Available’. at: 4.1.
208 <http://www.w3.org/TR> *Markup Language (XML)*, 2000/2001. (Web Services)