Chapter 4

SOAP-based Web services Discovery, Selection and Composition

In this chapter, the proposed framework and approaches for service discovery, selection and composition and experimental work of SOAP-based Web services have been presented. Section 4.1 discusses proposed framework for SOAP-based services discovery, selection and composition; Section 4.2 presents SOAP-based Web Service Composition Problem; Section 4.3 presents Web Service Matchmaking Algorithm; Section 4.4 presents Web Service Selection; Section 4.5 presents Experimental Work and Results; Section 4.6 presents Prototype for Healthcare Information System; Section 4 concludes the chapter.

1 Part of this chapter has been published as Kirit Modi, Sanjay Garg, An Approach for Ontology based Web Services Composition, Intelligent Systems and Control (ISCO2011), 2011 5th International Conference on, 2011.


3 Part of this chapter has been published as Sanjay Garg, Kirit Modi Sanjay Chaudhary A QoS-aware Approach for Runtime Discovery, Selection and Composition of Semantic Web Services, International Journal of Web Information Systems(IJWIS), Vol. 12 Iss: 2, pp.177 - 200
4.1 Proposed framework for service discovery, selection and composition

The proposed framework represents the relationship among discovery, selection and composition tasks of Web services as shown in figure 4.1. The operation of the framework can be described as follows:

- A composition process begins, when a Service Requester requests some functionality from service. Service Requester provides input parameters of service, required output and Quality of Service constraints as part of composition request.

- Composition Engine starts to discover a set of services, that provide required output and fulfill QoS constraints imposed by Requester from Service Repository with the help of Matchmaker and Service Selection & Filtering module.

- Matchmaker module performs matchmaking operation between User’s Query and
Chapter 4. SOAP-based Web services Discovery, Selection and Composition

Services provided by Repository. Reasoner is used with Matchmaker for computing similarity measure of the concepts. Matchmaker provides matched services to the Service Selection & Filtering module for selection, sorting and filtering of candidate services for composition. QoS parameters, i.e. Throughput and Response Time are used for filtering purpose. The matchmaking process is performed using domain ontology.

- Composition Engine is based on Composition algorithm. It generates composition plans using candidate services and provides best composition solution as an executable composite service.

The proposed framework consists of following components.

- **Service Requester**: Service Requester may be a person or a computer requests some functionality as required output by providing inputs along with QoS constraints.

- **Service Composition Engine**: It is responsible to compose candidate services provided by the Matchmaker and Service Selection & Filtering module. Composition engine interacts with Requester as well as Execution Engine. The output of this component can be an integration plan, a workflow, or a reference to a composite web service.

- **Service Selection & Filtering Module**: Service Selection & Filtering module is responsible to select services which can be used as service compositions. After selection, services are filtered based on requester’s QoS constraints. Overall quality constraint provided by requester would be compared for overall quality score of composite services for filtering. The selection process correlates various types of service descriptions (such as semantic, syntactic, QoS) at runtime to reach the best composition of Web services that suits the user’s need.

- **Reasoner**: A Reasoner uses the Domain Ontology database to compute the similarity measure of the concepts for semantic matchmaking process. The Reasoner performs semantic match using concepts of input parameters and output parameters of Web service during the matchmaking process.
• **Semantic Matchmaker**: The semantic matchmaker is used to perform matchmaking between user request with the set of services available in Web service Repository. If the service is matched with the requested service then it passes the result to the service filtering module which contains list of matched services as per the user request.

• **Web Services Repository**: This repository contains the semantically annotated Web services which is connected with the matchmaking module so that the available Web services can be discovered from the repository. QoS information is associated with the Web services. Each Web service provides Response Time and Throughput as QoS parameters.

### 4.1.1 Comparison of SUPER framework with Our Approach

This section summarizes the comparison of SUPER framework with the proposed approach.

• The framework implemented in the SUPER (Semantics Utilized for Process management within and between EnteRprises) project [Hepp and Roman](Hepp and Roman) is similar to Our Approach in terms of key components and their functionalities [Karastoyanova et al.](Karastoyanova et al.).

• The major objective of SUPER was to bridge the gap between the business needs expressed by business people and the actual Information Technology (IT) infrastructures intended to support them, while also supporting in a more efficient way the reuse and automation of business processes. For this reason, it implements a semantic-based and context-aware framework platform that supports the management of business processes, through the use of Semantic Web services technologies.

• The final platform of SUPER architecture includes modules for the automated discovery, composition and execution of business process implementations. Furthermore, three use case scenarios were developed for the needs of the project, all based on the telecoms domain, covering the fields of fixed telephony, traffic routing and
the management of mobile environments. Despite the different objective of SUPER project, it shares similarities with respect to Discovery and Composition modules in comparison with the proposed approach.

- SUPER project includes heuristic-based forward search AI planning approach, while I have used Greedy search approach for composition. The major drawback of AI-planning based approach is that they do not perform well in situations where the number of participating web services is large whereas Greedy search approach provides efficient solution for various levels of services within reasonable time.

- Information related to the non-functional properties (QoS) is not provided by the SUPER project whereas, I have used Response Time and Throughput as QoS parameters in the proposed work.

In order to develop the composition algorithm, basic terms related to web service composition are defined as follows.

**Request** ($R_q$): A composition request $R_q$ consists of a set of provided inputs $R_{qin}$, a set of required outputs $R_{qout}$ and a set of QoS constraints $R_{qos}$ imposed by the requester.

**Domain Ontology** (Ont): A Domain Ontology Ont consists of a set of concepts used to describe the domain in which a group of services is inserted. These concepts are related to each other according to subsumption relation, i.e. a concept $c_1 \in Ont$ subsumes another concept $c_2 \in Ont$ if $c_1$ is a superclass of $c_2$; $c_2$ subsumes $c_1$ if $c_1$ is a subclass of $c_2$; $c_1$ and $c_2$ subsume each other if they are the same.

**Web Service** (S): A Web service $S$ is defined as $S_i$, $S_o$, where $S_i = S_1, S_2, ..., S_i$ is a finite set of required input concept and $S_o = S_1, S_2, ..., S_j$ is a finite set of provided output concept.

**Service Repository** (SR): A service repository SR consists of a set of available services. Each service $S \in SR$ is composed of a required inputs $S_i$, a set of provided outputs $S_o$, and a set of QoS criteria $S_{qos} = (q_1, v_1); (q_2, v_2); ..., (q_k, v_k)$, where $q_i$ ($i = 1, 2, ..., k$) is a quality criterion, $v_i$ is the value of the criterion $q_i$ provided by the service, and k is the total number of criteria provided.

**Service Composition** (SC): A service composition SC is a Directed Acyclic Graph (DAG) with vertices $C_v = S | S \in SR$ and edges $C_e = (u, v) | \forall u, v \in C_v \land \exists c_1 \in u_{out} \land \exists c_2 \in v_{in}$
c_2 \text{ subsumes } c_1

**Comparator function** ($C_{cmp}$): A comparator function ($C_{cmp}$) sort the service composition (SC) and indicating whether SC1 ($r < 0$) or SC2 ($r > 0$) should be expanded next.

**Sorted list of composition** (X): A sorted list of composition X contains set of compositions Sorted according to comparator function.

**Requesters’ QoS Constraints** ($R_{qos}$): a set of QoS constraints provided by Requester for desired services. $R_{qos} = (q_1, v_1); (q_2, v_2), \ldots, (q_k, v_k)$, where $q_i$ (i = 1, 2, ..., k) is a quality criterion, $v_i$ is the value of the criterion $q_i$ provided by the requester, and k is the total number of criteria provided.

**Semantic Similarity between Concepts:** The semantic similarity between concepts refers to the degree that the two concepts, i.e. outR (a required concept) and outS (a service concept) can match, and it is a quantitative definition given on the basis of the four match types (Bellur and Kulkarni). The calculation formula is as follows.

- **Exact:** If outR and outS are exactly the same concepts;

- **Plug-in:** If outR is subsumed by outS, means outR is a subclass of outS;

- **Subsumes:** If outR is subsumes outS, means outR is a superclass of outS;

- **Fail:** when subsumption relationship between outR and outS is not available.

An algorithm for Web service composition shown in Algorithm [1] is inspired by the work proposed in (de Oliveira Jr and de Oliveira). The Algorithm [1] performs a service composition for a given request $R_q$. It starts by searching all services which can provide as output a concept which is semantically equivalent to the required outputs specified in $outR \in R_{qout}$. The matchmaking process is performed to calculate semantic similarity measure for a given request against available services. Matched results will be used for selection of candidate services (lines 2-8). A service will be a considered as candidate composition if it satisfies QoS constraints specified by user request $R_{qos}$. Quality parameters $Q_{qos}$ are extracted from the services and compared with quality constraints imposed in the request to get overall value of QoS. Candidate compositions are filtered those no longer
Algorithm 1 Web Services Composition Algorithm

Input: $R_{qout} \rightarrow \text{CompositionRequest}, SR \rightarrow \text{ServiceRepository}$

Data: $X \rightarrow \text{thesetofcandidatecompositions}$

Output: $SC \rightarrow \text{thecomposition, or } \Phi$

1: for each $outR \in R_{qout}$ do
2:  for each $S \in SR$ do
3:   $S \leftarrow \text{matchmaking}(outR, outS)$;
4:   $SL \leftarrow \text{selection}(S, Rqos)$
5:   $SC_v \leftarrow SL$;
6:  end for
7: end for
8: for Each $S_{qos} \in SC$ do
9:   if $\text{OverallQos}(S_{qos}) > \text{OverallQos}(R_{qos})$ then
10:      $X \leftarrow \text{append}(X, SC)$;
11:   end if
12: end for
13: while $X \neq \Phi$ do
14:   $X \leftarrow \text{sortServices(descending, X, C_{cmp})}$;
15:   $SC \leftarrow \text{popLastComposition}(X)$;
16:   if isSolution($SC$) then
17:      return $SC$;
18:   end if
19:   for each $S \in SR$ do
20:      $S \leftarrow \text{matchmaking}(outR, outS)$;
21:      $SL \leftarrow \text{selection}(S, Rqos)$
22:      $SC_{vnew} \leftarrow SC_vUSL$;
23:      $SC_{enew} \leftarrow SC_{eU} \forall S_2 \in cv(S, S_2)$;
24:      if $\text{OverallQos}(S_{qos}) > \text{OverallQos}(R_{qos})$ then
25:         $X \leftarrow \text{append}(X, SC_{enew})$;
26:      end if
27:   end for
28: end while
29: return $\Phi$;
meet the quality constraints (lines 9-13). The filtered candidate compositions are appended into X.

The candidate compositions are sorted in descending order using comparator function which compares two candidate compositions SC1 and SC2 and return a value \( r \) below zero if SC2 is more promising than SC1, above zero if SC1 is more promising than SC2 and zero if both are equally evaluated. The comparator function proposed in (Weise et al.) considers four composition properties: (i) known concepts-consists of input concepts provided by the requester and output concepts provided by the output of all services in SC; (ii) unknown concepts-consists of output concepts of the requester and input concepts of each service; (iii) eliminated concepts-consists of already provided unknown concepts; and (iv) the number of services in composition.

The comparator function considers overall QoS along with other four composition properties. Once the list is sorted using the comparator function, the composition is returned as a solution which has high overall QoS score (lines 14-19). Otherwise, the chosen composition is expanded to form new candidate compositions (lines 20-28). The algorithm runs until a candidate solution is found or all candidate solutions are expanded and rejected.

4.1.2 Algorithm Analysis

This section provides time complexity of the algorithm as follows. Time = \( O(p^3) \), Where, \( p \) is the number of services participating in the composition. The algorithm proposed (de Oliveira Jr and de Oliveira) have shown time and memory complexity \( O(p^m) \), where \( p \) is the maximum number of services participating in the composition (the number of services that can produce any concept required by the candidate composition) and \( m \) is the maximum number of services in a composition. Similar time complexity is achieved for this proposed approach.

In the best case, the time complexity of my approach is \( O(p^2) \), it defines the situation when composition solution will be found at first attempt. In the worst case and average case, the time complexity is \( O(p^3) \). It define the situation, when all possible candidate services participating in the composition required to be considered.
4.2 Web Service Matchmaking Algorithm

The matchmaking algorithm presented below is proposed by Choi et al. (Choi, Han, and Abraham) and used in (Bellur and Kulkarni). The same algorithm is used here as a matchmaking process as shown in Algorithm 2.

Algorithm 2 Web Services Matchmaking Algorithm

Input: outR (required concept), outS (published concept)

Output: Exact or Plug-in or Subsumes or Fail

1: if $outR = outS$ then
2:    return Exact;
3: else
4:    if $outS$ subsumes $outR$ then
5:       return Plugin;
6:    else
7:      if $outR$ subsumes $outS$ then
8:         return Subsumes;
9:      else
10:     return Fail;
11:    end if
12:  end if
13: end if

For the matchmaking Algorithm 2, the best case, worst case and average case time complexity is $O(1)$. It shows constant value for the time complexity.

4.3 Web Service Selection Algorithm

Set of Discovered services (SD): Represents services retrieved through discovery process.

Users criteria (Cj): Defines a criteria specified by the user through request. Normalized value of $QoS(N_{qos})$: Provides normalized value of QoS parameters.
Algorithm 3 Web Services Selection Algorithm

Input: $R_{qos}$ (User Request), SD (Set of Discovered services)

Output: SL (Sorted list of Selected services)

1: Begin
2: $S \rightarrow S_{qos}$;
3: for each $Si \in SD$ do
4: \quad if overall $S_{qos} > R_{qos}$ then
5: \quad \quad end if
6: end for
7: while $i \leq n$ do
8: \quad for $j = 1to2$ do
9: \quad \quad if $qi(Si) < Cj$ then
10: \quad \quad \quad filteroutSi;
11: \quad \quad end if
12: \quad end for
13: end while
14: for each $Si \in SD$ do
15: \quad calculate $C_Sp$ & $C_Sn$;
16: \quad $Q_{score} = N_{qos} + w$ ;
17: \quad $SL \lt sort(Q_{score}, desc)$ ;
18: end for
19: return SL
20: End
In the Algorithm\footnote{3} line 2 to 5 represents the list of candidate services for requesters with their QoS values. Line 6 to 11 represents the filtering mechanism. During the filtering process, matching of the user request criterion (cj) with each service criterion (qij) is performed. The services, which are unable to satisfy the requester’s specified constraints will be filtered out. The aim of the service filtering process is to generate refine result to the composition module to increase the quality and performance of the composite result. Line 12 to 15 represents ranking mechanism and select the best services according to highest QoS score. The ranking process is performed as follows. First of all scaling method is applied to get normalize value of all QoS parameters for each service then after the weighted value (w) is calculated. The total QoS score will be calculated by combining the normalized value and weighted value. Once the total score is computed, sort the services in descending order according the total QoS score. The selected list of services will be generated as result of the service selection process.

### 4.4 Experimental Work and Results

This section describes the detail about the experimental work carried out and results, which are generated based on the experiments. All the experiments presented here have been performed on with Intel Core 2 Duo 2.0 GHz processor with 3.0 GB RAM memory, Eclipse Europa with JDK 1.6.0, protége 4.3, Jena API.

In order to evaluate the proposed approach, I have used the Web Service Challenge (WSC 2009) dataset, which is a collection of five different datasets of semantically enabled Web services scaled from, 136 to 8016. These Two QoS parameters, such as response time and throughput are associated with these services to represent the quality aspects demanded by the user. In the following experimentation, the effect of increasing the size of Web services on the performance of proposed work is presented.

I have measured the execution time of algorithms by running them at least 5 times validation of results and get the average value of them. A sample query could be: response time≤1000ms, throughput ≥ 25. Then the query vector is set as: (1000, 25). A sample preference vector for this query could be: (1, 2).
4.4.1 Performance Parameters

Following performance parameters have been considered to measure the performance of the approach

(i) **Number of Services:** It defines total number of services available in the repository.

(ii) **Execution Time:** It represents the time required to perform the composition process.

4.4.2 Comparison of Proposed Approach with QoS-based Approach

A comparison of results of proposed approach with QoS-based approach (de Oliveira Jr and de Oliveira) is depicted in Figure 4.2. Results show that proposed approach performs better in comparison with existing QoS-based approach.

![Comparison of Proposed Approach with QoS-based Approach](image)

Figure 4.2: Comparison of Proposed Approach with QoS-based Approach

The results show that for different values of services of dataset the proposed approach performs well in comparison with the QoS-based approach, which has focused on composition process only while to make an approach more efficient integrated approach is used in the proposed work.
4.4.3 Comparison of Proposed Approach with Hybrid Approach

A comparison of proposed approach with Hybrid approach (Ma, Wang, and Zhang) is presented in Figure 4.3.

![Comparison of Proposed Approach with Hybrid Approach](image)

Figure 4.3: Comparison of Proposed Approach with Hybrid Approach

The Hybrid approach is an integration of Genetic programming technique with Greedy search method, which increases execution time of composition process while my approach is based on greedy search technique by integrating discovery, selection and composition process, which performs composition process in reasonable time and the generated solution is found to be near to optimal solution.

The following section discusses the prototype of Healthcare Information System (HIS). The purpose of this prototype is to validate the proposed framework. With this prototype, the applicability of the proposed work in the healthcare sector is presented. A comparison with existing prototype is also presented.

4.5 Prototype for Healthcare Information System

A Healthcare Information System (HIS) integrates the healthcare’s business process, and information systems to deliver better healthcare services[1]. The Healthcare information is offered through electronic medium, i.e. EHR (Electronic Health Record) but different
organizations uses different formats, which create problems of standardization and interoperability of the information. Some key challenges of EHR adoption are cost, ownership, standards and human factors. An ideal solution is required to make healthcare information interoperable and easily accessible by the end users. Another requirement is sharing of information for better and quick solution. By considering these challenges, I have developed a prototype for Healthcare Information System using the proposed approach as shown in Figure 4.4. The main components of the prototype framework are: discovery engine, selection & filtering engine and composition engine. At first, users' requirements are translated into XML based format using request formulation module and forwarded the request to the discovery engine.

The Discovery engine will perform semantic matchmaking based on user's request and semantically enabled services derived from the web service repository. Discovered services will be utilized for selection and filtering based on user specified QoS constrains. The resultant services are composed through composition engine and composite services

Figure 4.4: A prototype for Healthcare Information System
will be delivered to the user as a result.

4.5.1 Web Services for the Healthcare Information System

In order to describe the dynamics of the prototype framework, a use case from the electronic-healthcare domain is presented here. I define several Web services as shown in Table 4.1 related to Healthcare sector. List of Web services as shown in Table 4.1 contains semantically enabled services whose input and output parameters are mapped with concepts of the domain ontology of dentistry domain of Figure 4.5

Table 4.1: List of Web services

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Functionality of the Web service</th>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>getHospitalWithDoctor</td>
<td>Symptoms, Location</td>
<td>Hospital, Doctor</td>
</tr>
<tr>
<td>2.</td>
<td>getTransportation</td>
<td>Pickup-point, Landmark</td>
<td>Transport facility</td>
</tr>
<tr>
<td>3.</td>
<td>getAppointment</td>
<td>Date, Time</td>
<td>Appointment</td>
</tr>
<tr>
<td>4.</td>
<td>getPackage</td>
<td>Consulting Fee</td>
<td>Payment</td>
</tr>
<tr>
<td>5.</td>
<td>getOffer</td>
<td>Offer inquiry</td>
<td>Offer detail</td>
</tr>
</tbody>
</table>

The Web services such as getHospitalWithDoctor finds the nearest hospital at given location and a doctor with his/her expertise, getTransportation finds the transportation related detail, getAppointment retrieves the appointment detail and getPackage finds the payment related options.

I have added semantic description to these Web services using the ontology developed. These semantically enabled services are stored into the service repository for the discovery, selection and composition. Based on the knowledge-base, I define a user request that results towards the composition solution to fulfill the request.
4.5.2 Ontology Model for Healthcare Information System

The sample ontology presented in Figure 4.5 is the Dentistry Ontology, described using OWL and developed using prot´ég´e 4.3. Various subclasses such as Symptoms, Doctor, and Hospital can be described along with their properties and relationship with each other in the form of concepts. The sample ontology us used as the knowledge base.

4.5.3 Composition Process for Healthcare Information System

The depicts The conceptual representation of dependency relationship among participant medical services to generate a composition plan based on the user input is depicted in Figure 4.6. This composition plan shows that the user will provide the input only once, based on that the composition solution would be generated.

4.5.4 Performance of Healthcare Information System

I have utilized the two QoS constraints, such as response time and throughput as a part of the datasets.

The performance is measured by varying the number of services in the range of 100
upto 500 service for the purpose of comparison with DynamiCoS as shown in Figure 4.7. The obtained measurements show that the execution time of my prototype increases along with the number of services and show improved performance in comparison with DynamiCoS. At this juncture, it is important to note that the DynamiCoS approach has not considered non-functional parameters to perform service discovery, selection and composition tasks.
4.5.5 Comparison with Existing System

The Table 5.5 shows comparison of the proposed prototype with a Healthcare Information System ([Wang et al.] and Hospital application (Da Silva, Pires, and Van Sinderen) developed for medical appointment scenario.

<table>
<thead>
<tr>
<th>Aspects</th>
<th>Dynamic Healthcare Service Composition</th>
<th>DynamiCoS</th>
<th>Proposed Prototype</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic Web</td>
<td>HL7 Ontology</td>
<td>Domain Ontology</td>
<td>Domain Ontology</td>
</tr>
<tr>
<td>Information provided</td>
<td>-Information about the patient within organization -Nurses obtain patient’s report -Doctors prescribe treatment</td>
<td>-Provides Hospital location information only - Makes an appointment with doctor</td>
<td>-Provides Hospital Information -Provides appointment with doctor -Provides transportation detail -Offers package detail</td>
</tr>
<tr>
<td>Output</td>
<td>-Patient’s current report -Treatment given by the doctor</td>
<td>-Appointment detail only</td>
<td>-Appointment schedule -Hospital Detail -Doctor profile - Detail of Treatment charges</td>
</tr>
<tr>
<td>Repository-Size</td>
<td>Not specified</td>
<td>500</td>
<td>1000</td>
</tr>
<tr>
<td>Comparison</td>
<td>Not given</td>
<td>Not given</td>
<td>Provided</td>
</tr>
<tr>
<td>QoS Support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

4.6 Discussions

In this chapter, I have presented a framework for an integrated approach of service discovery, service selection and service composition for SOAP-based Web services by considering Semantic web and QoS-parameters such as response time and throughput. I have proposed the approach to realize the framework using greedy search technique. The proposed
approach has been evaluated using standard WSC datasets and compared with existing approaches of composition. I have demonstrated the application of the approach by developing the prototype for Healthcare Information System by using real Web services. The evaluation work and prototype implementation presents the efficiency and performance of the proposed work.