Constraint Based Clustering Technique for Web Services

Sumathi Pawar(✉), Manjula Gururaj, Roopa, Nirajan N. Chiplunkar, and RajaLaxmi Samaga

Nitte (Deemed to be University), NMAMIT, Karkala, Karnataka, India
pawarsumathi@gmail.com

Abstract. Web services are loosely coupled methods which need to be located and used according to the requirement. Loose coupling nature of Web services made the system to integrate or combine constituent Web services of a standalone business. A Web service application that can compose the Web service from different organizations will have transactions that span the multiple services. It is difficult for an application to create a software component and use it if it is unlikely to be reused. Therefore, this system provides reusability feature that allowed composing the Web services and these composable services are integrated into an application. Service composing is an ability to combine Web Services into complex applications to get the coarser grained behavior. The Web services may contain more than one operation. These operations are required to be clustered according to the required constraints. This paper is focused on constraint-based clustering to group the Web services according to user request for faster access. Constraint based clustering is a technique to integrate Web services according to required constraints. The system performance is checked with real time available Web Services during dynamic composition. This is a novel approach of clustering due to dynamic technique of collecting data during run-time.

Keywords: Constraint based clustering · Web services · WSDL · Composite services

1 Introduction

Constraint based clustering is a process of grouping the service operations according to assumed conditions. Integration of operations is the process of connecting the operations according to the input requirements [16, 17]. To get large information through a flow, it is required to compose Web services. Service designers create services for independent reuse using bottom-up design. So single web service may not be sufficient to satisfy user requests. Therefore, integration or composition of these modular Web services/operations is necessary to develop an application for user requirement. For example "invoice processing" service is a complex service which uses a "bank service", a "customer service", and description of an item ordered. These services can be used separately or together as complex service. The interaction among these services should

I. Woungang et al. (Eds.): ANTIC 2022, CCIS 1797, pp. 89–107, 2023.
https://doi.org/10.1007/978-3-031-28180-8_7

take care about service parameters like pre-condition, post-condition, type of web service composition.

The core of the service composition is sequencing and correlation. If the system has to understand the step by step operations of the business, the user must provide these steps of operations and system must provide a technique to maintain information across operations invoked, which is called as correlation. To fully automate the creation of a composed Web Service application, the user need additional information such as service availability, response time and service quality information. It is difficult to get these information without service execution (no UDDI). In early days QoS information of the Web services was stored by UDDI. Because of the absence of UDDI it is possible to get QoS information such as availability, response time and security information only after execution of the services.

The existing systems used to pass the transaction context from service to service are Business Transaction Protocol (BTP) from OASIS, WS-Transaction from IBM and Microsoft and Activity Service from OMG that provide reliable outcomes. JTS and JTA transaction managers are used to manage and coordinate distributed, ACID transactions.

1.1 Web Service Composition

In this research, Web service composition is also referred as integration. Composition models are organized into three dimensions-orchestration, choreography and coordination. Composition language like WS-BPEL (Web Service Business Process Extraction Language) can be relatively translated onto classes of formalism like automata, Petri net, process alzebra and other formalisms.

But the proposed research is not using UDDI registry as the third party because of unavailability of public UDDI registries. [16, 17] Instead, this research is using Bingo search engine technique. The user of the proposed system gives the required functional word as a query to the search engine in the form "functional_word?wsdl" during run-time. Then the search engine returns various links of WSDL to related services. The input parameters of the operations of these WSDL are compared with user queried parameters and those having non-zero support value are considered.

1.2 Web Service Description Language

The WSDL describes the service which belongs to service description layer.

This layer describes the three aspects of the web services.

- Operations of a Web Service
- Messages of the service that accepts
- Binding protocols to access the service

Web services are described by the Web Service Description Language (WSDL) by which the service consumer is able to discover a set of useful operations. In this research, WSDL's abstract descriptions and concrete descriptions are processed to get service information for satisfying the user request or for integration of the service.

1.3 Location Transparency

According to the Web service concept, service environment achieves location transparency because the location of Web service is stored in a registry. But service registry UDDI is not public as specified in the literature review. In this research UDDI is not used. But location transparency is achieved by the Bingo search engine which finds the service even though the service is moved from one location to another. Bingo searches the WSDL of the service and even though service is moved from one plat-form to another, no changes to the client application is necessary.

1.4 Scalability and Availability

This system is scalable because at the client side, client only knows interface of the service and not its implementation. At the server side any number of services can be added without overhead at the client application. In this system, availability of the service is high because even though one service fails, other similar service's information are stored in the cluster and used for satisfying the user request.

1.5 Dynamic Web Service Discovery

The constraint based clustering is helpful in discovering Web services dynamically. Exiting researches were discovering Web services using UDDI. According to the survey conducted, it is practically checked that UDDI is absent since the year 2006. This motivated our research to use search engines. The search engine used in this system is Bingo. The search result retrieved from the Bingo gives discovered information of the Web services from all the servers in World Wide Web in the form of WSDL. The search query that can be given to the search engine is prepared according to the user given request during run time, which makes the system dynamic.

1.6 Dynamic Web Service Invocation

Clusters which stores the operations are helpful in dynamic service invocation which is done at run time by filling the required parameters for invocation of the Web service dynamically. The parameters need to be filled are according to the user requested function and matched parameters are extracted from suitable WSDL of the Web services and populated in the place of the unknown parameters. The entire process is automatically executed and input parameters are asked from the user if user knows it. In the proposed research, human interaction is required only to enter the input parameter's value if input is known to the user. Or else, this system automatically searches the unknown input by using available Web services. These values are searched automatically in the online World Wide Web to satisfy the user requests. This reduces the burden of the user filling the unknown parameters of the required Web services.

1.7 Automatic and Dynamic Web Service Composition or Integration

The proposed constraint algorithm is also helpful in integration of the Web services that involves the task of service search, service selection and the connecting required operations of the Web services. The connecting of different operations is done only when it is required to resolve the values of unknown parameters which are resulted into the Web service composition. Many existing researches did service composition using different techniques, which need some prior information about the selection of services. But in the proposed research, without any knowledge-base, Web services are composed or integrated according to the requirement of the user which makes the system fully dynamic. In the proposed system, to overcome dynamic nature of the Web services, search of composable service is performed automatically by using search engines to get the available online Web service operations through clusters.

1.8 Automatic Web Service Execution Monitoring

The proposed provides clusters with input parameters and operation names of Web services. In the existing researches, the automatic service execution is not resolved by filling the parameters dynamically. The monitoring of the service execution is necessary to get information about the availability of services and the response time. Now a day due to absence of UDDI, no source like UDDI gives information about the QoS of the services. This information are noted during execution of Web services and stored in the repository, so that these information are used in the future during the composition of the Web services. The failing of the service execution is also recorded, so that in future the same service is not used and instead other similar services may be used. The composition plan generated in this system is tested for availability and response time and gives QoS to future requests.

1.9 Scope of the System

The proposed system is implemented as client for using existing online Web services. This research is implemented for information retrievable online Web services and not for transactional Web services. Because transactional operations cannot be implemented without co-ordination between executions of operations and there are no such free transactional Web services.

There are very limited WSDL described Web services available in the online nowadays and most of these services are information retrievable Web services. Therefore scope of the system is information retrievable Web services and the present system is beneficial only when there are composable Web services available in the online. By considering the atomicity of the operations, Web services are maintaining fine grained level of information. Hence scope of this system is, also to get coarse grained information by dynamically composing the operations.

2 Literature Survey

[1] Authors proposed Ontology based context models for context aware applications. They provided a discovery algorithm which matches the requested word with service

Ontology and which gives best matched service to the user. Authors considered location, time, person or agent as key contexts and included service Ontology in Context broker architecture.

[2] Each incoming request randomly chooses the one *k* dominating services to respond to the request. The Web services selected by top-k dominating services reduce the computation space instead of selecting the best solution. When concurrent requests are given to the same service, response time of that service may reduce due to many reasons. But in this system as k increases, the time of retrieving top-k dominating service also increases. Hence the response time of this system also increases. Authors provided an algorithm for selection of top k dominating Web services by calculation of the scores. But the cost of calculating the score of the Web services depends upon the efficiency of the algorithm and in order to quickly obtain the top *k* dominating services, they aggregate R-tree (aRtree). But in the proposed research, selection of Web services depend on user opinion, and hence user interaction is very important in the selection of Web services.

[3] The quality of Service is a factor to enhance the efficiency of Web Services. Technical quality and managerial quality are 2 kinds of qualities of Web services. The operational aspects of Web services are called technical quality. Technical quality deals with performance, response time, reliability, latency, execution time, throughput, dependability, availability, reliability, failure semantics, failure masking, operation semantics, exception handling, compensation, robustness/ flexibility, capacity, scalability, security, continuous availability, compliance, reputation/positive feedback and network related QoS. The managerial qualities deal with management information such as ownership, contact, payment etc.

But no such QoS parameters are available nowadays for real-time Web services because of a permanent shutdown of UDDIs. Therefore the proposed system could not use QoS parameters, instead it gives user satisfaction as feedback to rank the Web services.

[4] Depending on the nearest location of the service, service invocation is done to minimize data transmission time and cost. They did it by first selecting the k number of abstract services' service library, and then from each group selecting the one candidate service. But they assumed that service is gathered as a set of services within one area and not randomly distributed. By considering the returning time of service result, they calculated the distance of the service from the consumer.

[5] Maintenance of RESTful Web service is easier than the maintenance of SOAP-WSDL Web services. RESTful web services communicate using XML files and HTTP-GET and SOAP-WSDL Web services communicate using HTTP-POST. Authors proved that maintenance of RESTful web services at server side is easier than SOAP-WSDL Web services because they are light weight.

[6] TESSI, a tool based on TASSA, is a solution for WSDL-based testing of both single and composite web services. Test cases are defined in the XML form and each test case has two parameters, test case name and test case template. TASSA carries out the tasks such as identification of service operations, the creation of SOAP request messages, the definition of assertions at BPEL variable level, execution of test cases with sending and receiving of SOAP messages and collection of test outputs for result analysis.

TASSA framework tests the Web services for not only functional correctness but also for performance. Black box testing for BPEL is also followed which checks end-point orchestrations. Execution of test cases provides details about the behavior of the service.

Isolation tool, Data Dependency Analysis tool, Value Generation Tool, Injection tool and Test Case Generation tool are different tools available in TASSA framework. But the tool is applied only during design time testing of service Oriented Applications.

[7] The network usage is better than WS-Management, because the size of SOAP request and response messages is larger in WS-Management than in SNMP. But in Web Service-based management middle-ware models they managed single service at a time and not considered complex composite services. In the proposed research, the network load will be reduced because of usage of MOLAP to handle big size of WSDL of single Web services.

BPMN [8] modelor and tactic models are created to specify different steps of the composition and each tactic is implemented by service operations. Discovering the requirements are affected by unknown context is handled by creation of rule. Observation of context and collection of context information is done by Evalution planners.

But creation of composition models is static in nature which does not handle the dynamic requirements without the knowledge of context. The proposed research handles dynamic requirement by dynamic searching of Web services and by generating dynamic composition plans.

[9] In a sensor network performance of REST is better compared to SOAP, and it is recommended by equipment vendors to use RESTful Webservices for applications consuming embedded resources. But the proposed system works on only information retrievable services which are able to provide secured data transfer for future works. Therefore this system is not using RESTful Web services.

[10] The combinatorial optimization problem is composed by many services to build Service Networks that satisfy many requirements based on cost-effectiveness. Execution Time, reliability and price of services are used as QoS parameters. The performance is measured based on single requirement and also based on multiple requirement. This system uses Service Network as a tool which gives number of connections between services and the QSC approaches, which provides global QoS performance. Performance of the system is measured with total benefit and execution time. But in the proposed research, service execution performance is increased using MOLAP data model.

Due to removal of some Web services and change of information such as change of transaction fees, the authors created the concept of [11] change management of long term composition [12]. The failure in distribution of reputation values is that, the component Web service is never be penalized for poor performance of other peer component Web services. This is done by reputation propagation by changing the behavior of the individual services and thus increasing the reputation of the composite Web services. They divided the service composition into horizontal composition, vertical composition and hybrid composition. Web service composition is done by an agency which distributes the reputation values fairly. Here reputation is consumers perception about Quality of Service such as performance, reliability and availability, about the service they invokes. But when the consumer invokes composite Web service, their perception will be about composite Web services and these perceptions will not be about component Web services. Therefore average of reputation value is computed and distributed to component Web services. Reduction of reputation values are handled by composition orchestrator and perform the decision in finding the malicious Web services. It is a very important role of orchestrator to make decision.

[13] Another important issue of QoS is Trust based management of Web service which is divided into different types-Policy-based trust, Reputation-based trust etc. Trust Metrics given for the Web services are Execution Time, Response Time, Latency, Availability, Reliability and Remedies. As given in the survey WS-POLICY is the policy document that provides the policy-information and reputation is based on user perception reputation of the Web services [14]. In existing system, term pairs are entered in the Google search engine and snippet retrieved from Google search engine is used to get association of words such as "apple computer", "hardware software" etc. They categorized those terms into particular contexts and then generated the context vectors by computing the TDF/IDF values. This context is used to calculate the service similarity.

In proposed system, after extracting the WSDL elements of Web services, the elements such as service name, operation name, input and output features are compared to the service elements of the cluster of the Web services. Precision, recall and F-Measure are the factors used to measure the performance of the system. In the proposed research, every component Web services' availability and performance time is recorded and do not depend on one centralized system. In the proposed system, QoS factors are recorded during each Web service composition and current values are considered for next transaction.

3 Clustering Web Services with Constraint-Based Clustering

In this algorithm the WSDL links of each requested functional words are given as input to the procedure named constraint Based Clustering. The operations of each service are compared with each other for exact, partial and synonym matching factors and stored in the appropriate clusters as given in the pseudo-code below.

```
Procedure constraintBasedClustering(RetrievedWSDL_Links)
Step 1: Initialize I to 0
Step 2: For each WSDL_Links of RetrievedWSDL_Links extract $i^{th}$ service name
    Step 3: Extract operation names for $i^{th}$ service
    Step 4: Keep the operations in the $i^{th}$ cluster : i=i+1
Endfor
Step 5: initialize j to 0
Step 6: For each $j^{th}$ operation of cluster
    Step 7 : Apply Decomposition Rules on $j^{th}$ operation and store functional part of operation name in sourceOp:
      k=0
    Step 8: for each $k^{th}$ operation of the cluster
        Step 9 : Apply Decomposition Rules on $k^{th}$ operation and store functional part of operation name in destOp
        Step 10 : If sourceOp and destOp exactly matches then
            Step 11 : store destOp in ExactMatchingCluster(j) : j=j+1
        Else
            Step 12: if sourceOp and destOp matches partially then
                Step 13: store destOp in partialMatchingClusters(i) : j=j+1
            else
                Step 14: Retrieve words similar to destOp from Wordnet and store it in SynonymWords
                Step 15: For each synonym of SynonymWords
                    Step 16: Compare sourceOp to synonym
                    Step 17: If matches then store destOp in synonymMatchingClusters(j) :j=j+1
Step 18 : Break :Endif: Endfor: Endif : Endif : k=k+1: Endfor: Endfor : End Procedure
```

Fig. 1. Constraint based clustering

The constraint-based clustering is algorithm shown in the Fig. 1. The flow of this clustering process is shown in the Fig. 2. In this clustering, the operation names of retrieved WSDL links of requested functional word are stored in the cluster C1 initially. After applying decomposing process to operation names of each operation, the clusters are formed as shown in the pseudo code of the Fig. 2.

As an example input to the proposed system, consider the user requested function is “Weather”. The different WSDL links retrieved for this request are given in the Table 1 of Sect. 5. Among the operations of the “GlobalWeather” service, the operations “GetWeather” and “GetCitiesByCountry” are to be tokenized using decomposition rule. The sets generated are S1 = {Get, Weather} and S2 = {Get, Cities}. In the operation “GetCitiesByCountry” the token after the stop word “By” is “Country”. This token is considered as input token and stored in the input set of this operation.

The requested function “Weather” is compared with the tokens of first set S1. If it matches exactly to any token, then the <operation> name belong to this set is stored in the exact matching cluster of this service. Otherwise if it matches partially, then the <operation> name belong to this set is stored in the partially matched clusters of this service.

If in both situations if the requested token does not match, then words similar to the <operation> name element is retrieved from the *Wordnet* dictionary and compared against the user requested functional word. If any of the synonym word matches to the user requested function exactly, then this <operation> element is stored in the synonym matching clusters of this service. Finally all the exact matching clusters are merged together. Similarly the partially matching and synonym matching clusters also merged.

In the “getWeather” operation, the “Weather” token of the set S1 is matched with the requested functional word “Weather” exactly. Since “GetWeather” operation is stored in the exact matching clusters. There are no matching tokens in the set S2 for the requested functional word.

Each operation name is *Decomposed* first. Care is taken when stop words are removed because the operation name “GetWeatherByZipcode” and “ZipcodeToCityState” are not similar operations even though it contain “zipcode” as common word. Therefore when tokenizing the operation names, the words like “By”, “To” are interpreted properly. The operation names which have tokens before “By” are interpreted as operation name and tokens which come after “By” are interpreted as input names. The names of the operations which come before the “To” are treated as input parameter name and parameters which come after the “To” are operation names. Therefore these two operations belong to different clusters.

The operations of other semantically matched service for “Weather” request are operations of “WeatherForecast” service i.e. “GetWeatherByZipCode” and “GetWeatherByPlaceName”. Here token sets of these operations are S1 = {“Get”, “Weather”} and S2 = {“Get”, “Weather”} respectively. These tokens exactly match with the requested functional word and stored in the exact matched clusters. Tokens after the stop word “By” are “ZipCode” and “PlaceName” are considered as sets of inputs of respective operations.

The flow of forming the cluster of operations of retrieved WSDL links are shown in the pseudo code of Fig. 2.

C1 = Cluster of Retrieved WSDL links of requested functional word.
CO(k) = kth Cluster of operations of services of kth WSDL link where k ranges from 1 to n and n is an element of finite set of natural number.
CEM(k) = kth Cluster of exactly matching operations to the functional word.
CPM(k) = kth Cluster of partially matching operations to the functional word.
CSM(k) = kth Cluster of synonym matching of operations of this service to requested functional word.

Here exact matching (CEM(k)), partial matching (CPM(k)) and synonym (CSM(k)) matching clusters are three constraint based clusters formed through this process.

The clustering process with score is shown in the Fig. 3.

At first each of the WSDL link is considered to extract operation names. Each operation name is decomposed and requested functional word is compared with decomposed tokens. The cluster of exactly matching services CEM(k) stores the operations which exactly matches to the requested functional word. The weight given to the tokens/operations matches exactly is 0.5. The partially matching operations to the functional word are given by weight 0.3 and stored in the cluster CPM(k). The synonym matched operations to requested functional words are stored in the cluster CSM(k) and weight given to synonym match is 0.2.

Initially all the operations of the service$_1$ to service$_n$ of retrieved WSDL links are stored in the cluster called C_1 to C_n Where n is the element of finite set of natural numbers.

Then requested functional word is compared with the decomposed operation names of C_k and if requested functional word matches exactly then that operation name will be stored in the CEM(k).

If Requested functional word does not match exactly to any operation of C_k then CPM(k) and CSM(k) are formed for partial matching and synonym matching where k is the element of finite set of natural numbers.

All exactly matching operations are merged into exact matching clusters. This CEM cluster gives all exactly matching operations to the requested functional word. All synonym matching operations are merged into synonym matching clusters and all partial matching clusters are merged into partial matching clusters. If exactly matching clusters are empty for a functional word then partial matching clusters are considered. If partial matching clusters are also empty then synonym matching clusters are considered. If all the clusters are empty then it is concluded that requested functional words does not exist as Web services in the online.

During selection of matched operations, if no operation is matched exactly then the partial matching operations are retrieved from already framed partial matching clusters. If partial matching clusters are empty then operations from synonym matching clusters are retrieved.

3.1 Clustering for Integration of Web Services

Some of the operations of a service can form a cluster because these operations are linked to each other and are integrated according to the user requirement. Different types of clusters formed are given in the following sections.

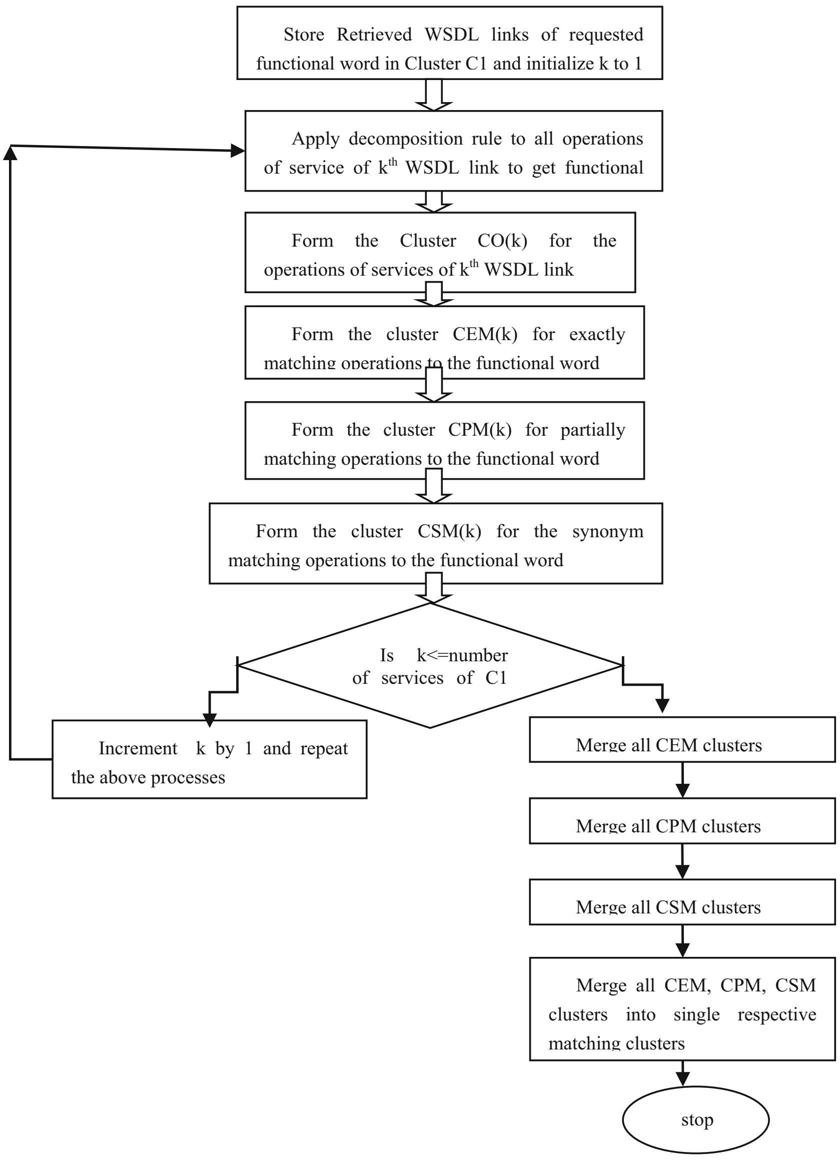

Store Retrieved WSDL links of requested functional word in Cluster C1 and initialize k to 1
Apply decomposition rule to all operations of service of kth WSDL link to get functional
Form the Cluster CO(k) for the operations of services of kth WSDL link
Form the cluster CEM(k) for exactly matching operations to the functional word
Form the cluster CPM(k) for partially matching operations to the functional word
Form the cluster CSM(k) for the synonym matching operations to the functional word
Is k<=number of services of C1
Increment k by 1 and repeat the above processes
Merge all CEM clusters
Merge all CPM clusters
Merge all CSM clusters
Merge all CEM, CPM, CSM clusters into single respective matching clusters
stop

Fig. 2. Flow of process of clustering of operations which matches functional word through constraint-based clustering.

3.1.1 Self link Clusters

One of the constraint used in this research is, all operations of a service which are supplement to each other are belong to the same cluster. This is shown in Table 2 where all operations of a particular Web service are given together. For example in

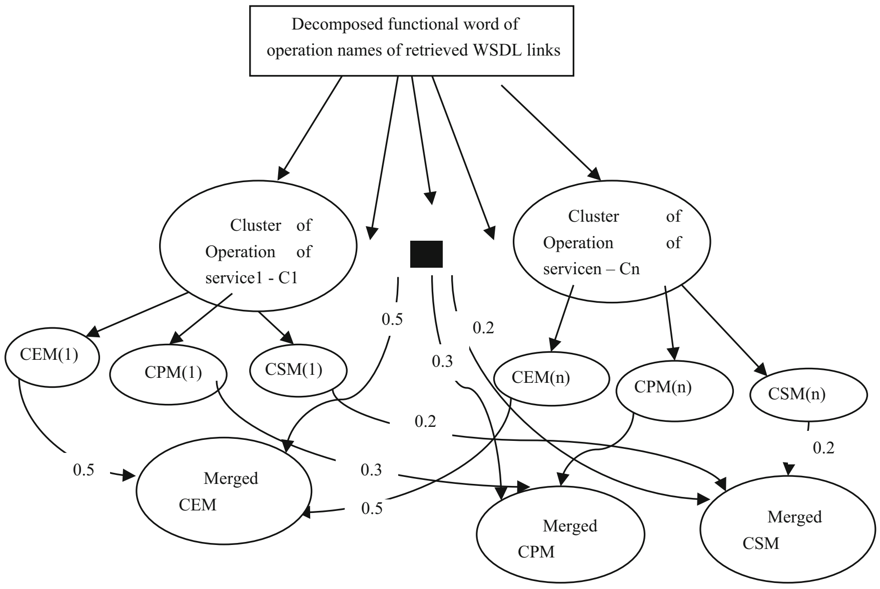

Decomposed functional word of operation names of retrieved WSDL links
Cluster of Operation of service1 - C1
Cluster of Operation of servicen – Cn
CEM(1)
CPM(1)
CSM(1)
CEM(n)
CPM(n)
CSM(n)
Merged CEM
Merged CPM
Merged CSM
0.5
0.2
0.3
0.2
0.5
0.3
0.5
0.2

Fig. 3. Exact, partial and synonym matched constraint based Clustering

"GlobalWeather" service the operation "GetWeather" and "GetCitiesByCountry" are related because for GetWeather operation inputs are "CityName" and "CountryName".

To get information about names of cities of different countries it is required to invoke the operation "GetCitiesByCountry". Therefore both "GetWeather" and "GetCitiesByCountry" will be belonging to one link-based cluster. This is justified by comparing input elements of "GetWeather" with <operation>/<output> elements of "GetCitiesByCountry".

Each input element of the all the operations of the required service are compared with <operation>/<output> elements of all the remaining operations of the same service. Clusters will be framed for operations which are supplement to each other within the same service for getting the value of input of operations. This type of clustering is called *self-link clustering*. In the self link clustering the input of particular operation is satisfied by output of other operations of same services (Fig. 4).

3.1.2 Output-to-Input Linked Clustering

The output-input linked clusters contain operations which are linked together. The services which require to get the input from the output of operations of other services are gathered into a cluster called output-input linked clusters. Comparing the input elements to the <operation>/<output> elements is done by exact matching function.

In the Fig. 5 the following abbreviations used.

CS – Composit Service.
CS1 – "Input1 of the Composit Service".
CS2 – "Input2 of the Composit Service".

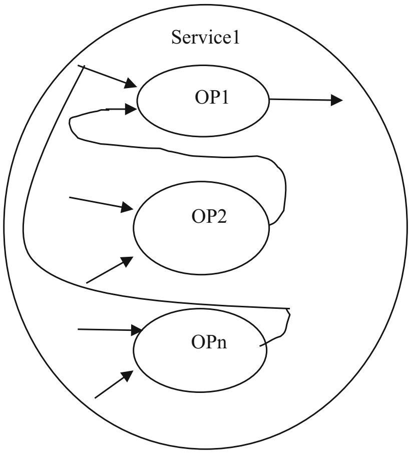


Fig. 4. Self-link clusters

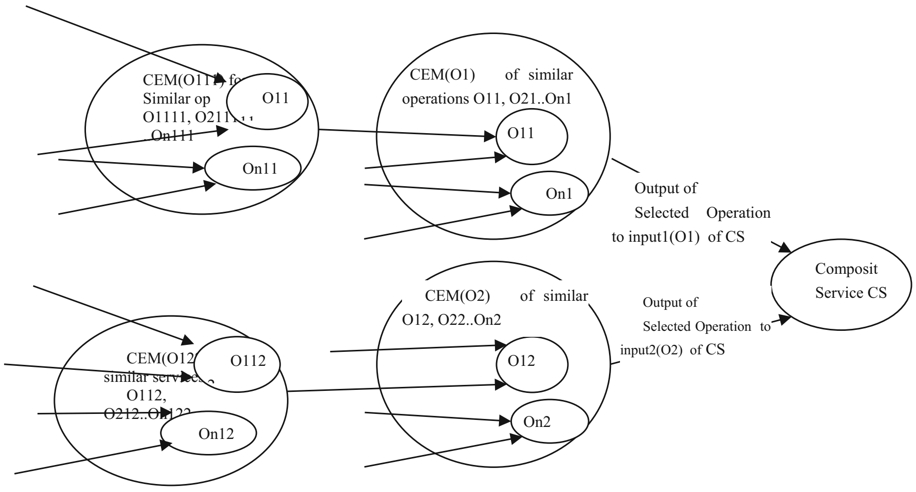

CEM(O111) for Similar op O1111, O2111..On111
O11
On11
CEM(O1) of similar operations O11, O21..On1
O11
On1
Output of Selected Operation to input1(O1) of CS
Composit Service CS
CEM(O2) of similar O12, O22..On2
O12
On2
Output of Selected Operation to input2(O2) of CS
CEM(O122) similar services O112, O212..On122
O112
On12

Fig. 5. Output to input linked clusters

CEM(O1) – “Cluster of exact matching operations which has matching output to the input1 of composit service”.
CEM(O2) – “Cluster of exact matching operations which has matching output to the input2 of composit service”.
CEM(O111) – “Cluster of similar operations which matches input1 of operation O11”.
O1 – “Operation1 which matches input1 of CS”.
O2 – “Operation2 which matches input2 of CS”.
O1111 – “Operation1 which matches input1 of the operation11”.
On111 – “Operation which matches input1 of the operation11”.
O1122 – “Operation1 which matches input2 of the operation12”.
On122 – “Operation which matches input2 of the operation12”.

The above Fig. 5 shows the output-input link based clusters which have similar services of different community. Consider the composite service CS which has unknown input value for both input1 and input2. When there is a search for input1 it is called as composit search1 (CS1) for input1 and when there is a search for input2, then it is called as composit search2 (CS2) for input2.

In the Fig. 5 which shows output-input link based clustering, the similar operations which match the input1 of CS (complex service), form the cluster CEM (O1). From this cluster, suitable operation is selected by using weight matrix of WSDL. The cluster CEM (O1) gives similar operations O11, O21…On1 which matches the input1 of complex service CS. The value of another input of complex service (CS2) is also unknown. The cluster CEM (O2) includes similar operations O12, O22..On2 (n operations) which have matching output to the input2 of complex service(CS). The cluster of similar operations CEM (O111) gives matching operations which have the output that matches to the name of input1 of the operation O11.

4 Implementation

This research is implemented by using existing Web services in the online. The Web services can also be created in the local server. To create the Web services in the local server the Tomcat server is used in the proposed system. In one Tomcat instance, one service is started. Number of Web services is kept in the run mode in different instances of Tomcat Web server. Each Web service is running in the one instance of the Tomcat. But because of bottleneck of the capacity of the server, thousands of Web services cannot be kept in the run-mode simultaneously. For this purpose number of servers are required. Therefore it is recommended to use available online Web services. But available online Web services consume enough bandwidth to retrieve WSDL and more storage space to process and store the WSDL elements.

This research is implemented in Java Eclipse framework with 8 GB Ram and i5 processor. Code is written and executed for real time Web services. To store the operation elements of WSDL in the, it is required to extract operation elements from the WSDL. The <operation> element is extracted from the WSDL using.

WSDLHelper.GetOperations(portTypes) statement.

The portTypes is the element of WSDL which contains set of operation elements. The portTypes is extracted by using the statement.

portTypes = WSDLHelper.getPortTypes(WSDLdefinition).

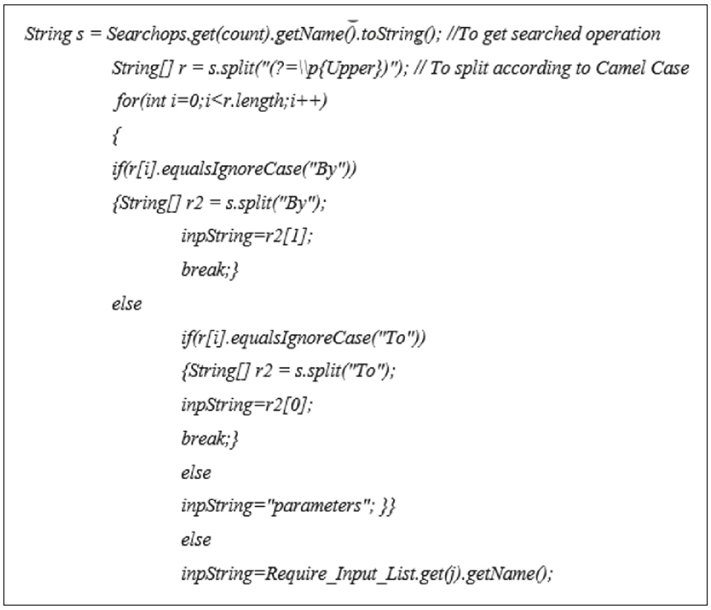
Each operation is to be iterated and stored in the cluster using operation_keylevel2[cnt][count] = Searchops.get(count).getName() statement.

The input parameters are stored using the statement.

inputlevel3[cnt][count][j] = WSDLHelper.getInMessageParts(Searchops.get (count)).

The above statement stores the input parameters of the searched operations. If the input parameters store correct parameter names, then it is helpful to the user to enter the value of input parameters. If the input parameter names are given as “parameters” in WSDL, then the real input parameters’ names are obtained by mining same operation name’s binding information. It is already observed from the WSDL data set that, unknown input parameters can be mined by different binding information of the same operation.

Code snippet to interpret the input parameter names from the <operation> element



```
String s = Searchops.get(count).getName().toString(); //To get searched operation
            String[] r = s.split("(?=\\p{Upper})"); // To split according to Camel Case
            for(int i=0;i<r.length;i++)
            {
            if(r[i].equalsIgnoreCase("By"))
            {String[] r2 = s.split("By");
                      inpString=r2[1];
                      break;}
            else
                      if(r[i].equalsIgnoreCase("To"))
                      {String[] r2 = s.split("To");
                      inpString=r2[0];
                      break;}
                      else
                      inpString="parameters"; }}
                      else
                      inpString=Require_Input_List.get(j).getName();
```

Fig. 6. Code snippet to extract input parameters from operation names of the Web servic

This code snippet of Fig. 6 splits the operation name according to the camel case. For example if the operation name is "GetWeatherByZipcode" then it is splitted as Get Weather By Zipcode. If the operation name is "CityStateToZipcode" then it is splitted as City State To Zipcode. Here the word after "By" and word before the "To" are input parameter. This word is stored in the place of unknown parameter name.

As a search result, Web services with "UsZip" and "AddressLookup" services are received as results. In this "UsZip" service is matched partially to the "zipcode". Therefore the operations of this service "GetInfoByAreaCode", "GetInfoByZIP", "GetInfoByCity" and "GetInfoByState" are listed and shown to the user. Among these operations if the user selects "GetInfoByZIP" then the input parameter "UsZip" is unknown to the user. Therefore, this operation cannot give required output and the user has to select "GetInfoByCity" where the output parameter is "Zipcode" and input parameter is "City". This operation matches with the unknown element because the search technique found the unknown input in the output of this operation element.

When the operation selected by the user is "GetInfoByCity", then the input parameter of this service "UsCity" is shown to the user and user is allowed to enter the city name. If the city name is entered by the user then the system invokes "GetInfoByCity" operation of the "USZip" service and returns the result. The returned results are very large and are in the XML form which has more than 300 rows of zipcodes of different areas. But there will be a deserialization problem in getting these results because some of the XML data types of these results cannot be deserialized by the java data types. Therefore invocation of "UsZip" service results in the exception and fails in returning results. Hence the query for the same service with populated operation name and input parameters are given as URL in the browser in the following form.

URL = http://www.webservicex.net/uszip.asmx/GetInfoByCity?UsCity=Washington

At first the above query is stored in the URL object and given through the J2EE API as

BufferedReader in = new BufferedReader(new InputStreamReader (URL.openStream()));

The above method creates the buffer to store the results and XML results will be processed to extract suitable data.

5 Results

The below Table 1 shows the links retrieved for the request and returned results' operation names are clustered using constraint-based clustering technique. Support and confidence of search precision is calculated.

Table 1. Links of retrieved WSDL links for request Weather?wsdl

Requested word	Service name	Retrieved WSDL links
Weather	GlobalWeather	http://www.webservicex.com/globalweather.asmx?WSDL
	TemperatureConversions	http://webservices.daehosting.com/services/TemperatureConversions.wso?WSDL
	WeatherForecast	http://www.webservicex.net/WeatherForecast.asmx?WSDL
	CurrencyConvertor	http://www.webservicex.net/CurrencyConvertor.asmx?WSDL
	Service	http://www.ejse.com/WeatherService/Service.asmx?WSDL
	Service1	http://www.tempe.gov/wx/Default.asmx?WSDL

Figure 7 shows the graph of support(S) value of integrated/composed web services to give result of complex web services. Here request given by user is weather of a country of unknown input parameter value. Then this unknown input parameter will be resolved by the system with the help of constraint-based clustering. The result gives 40%, 60% and 40% support values with the real time web services for different composition plans. Support and confidence value of composition is increased if there is a greater number of similar services available during runtime.

Table 2. Cluster of Operations and input parameters of retrieved WSDL links for request Weather?wsdl

Service key	Operation name	Input parameters
Weather	Get_Weather	Parameters
	Get_Cities_By_Country	Parameters
	Get_Weather	CountryName CityName
	Get_Cities_By_Country	Country
	Get_Weather	CountryName CityName
	Get_Cities_By_Country	Country
WeatherForecast	GetWeatherbyPlaceName GetWeatherbyPlaceName GetWeatherbyPlaceName GetWeatherbyZipcode GetWeatherbyZipcode GetWeatherbyZipcode	Parameters PlaceName PlaceName Parameters Zipcode Zipcode

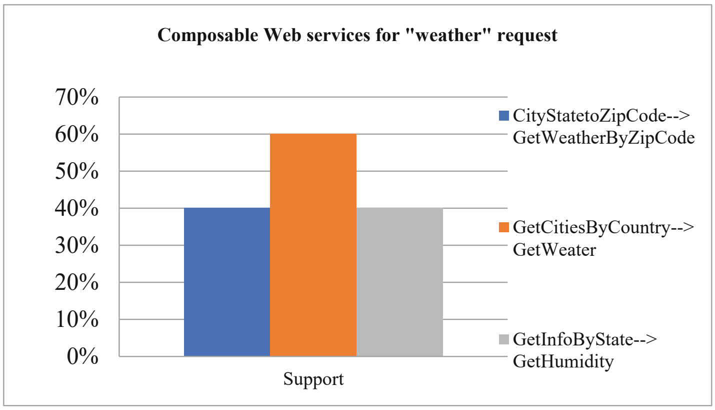


Fig. 7. Support of multiple composition plans using constraint based clustering

Precision, recall and f-measure are calculated using Eqs. 4, 5 and 6. As shown in the graph of Fig. 8, performance is calculated in percentage values. Precision is 58%, recall is 59% and F-measure is 52% for a particular request. This performance varies for different requested Web services according to user requirement.

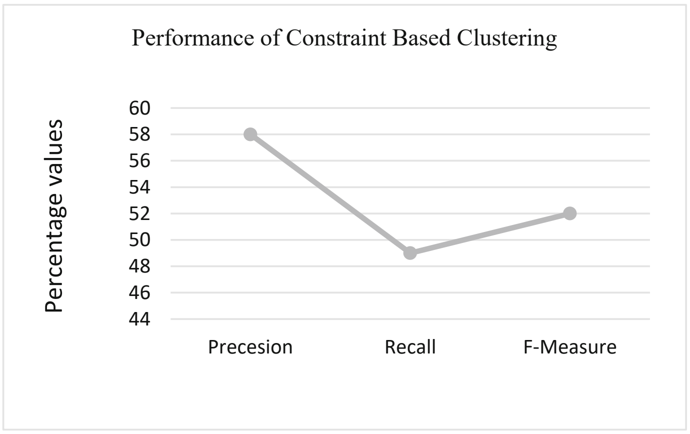


Fig. 8. Performance of constraint based clustering

6 Analysis

In this research using constraint-based clustering, service composition is achieved with following performance factors.

6.1 Support and Confidence of Search Precision

To measure the precision of WSDL retrieval, support and confidence factors are used. The precision is given with the following equation

$$\text{Precision} = \frac{Retrieved \cap relevant}{retrieved} \tag{1}$$

The search precision of retrieved matched WSDL from the Bingo search engine is calculated as dividing the count of relevant WSDL by count of retrieved WSDL.

In the same way support and confidence of matched operation in the retrieved Web services are calculated in Eq. 2 and Eq. 3 as

$$\text{Support} = \frac{\text{count of matched term in any element of WSDL}}{\text{count of retrieved WSDL}} \tag{2}$$

$$\text{Confidence} = \frac{\text{Count of requested functionality in operation element}}{\text{total number of operations}} \tag{3}$$

$$\text{Recall} = \frac{Retrieved \cap relevant}{relevant} \tag{4}$$

$$\text{F-Measure} = \frac{2{*}Precision * Recall}{(Precision + Recall)} \tag{5}$$

Performance of this system varies according to real time available services because this is dynamic real time Web service clustering system.

Equation 6 gives support(S) of multiple composition plans [1] of real time Web services using constraint based clustering

$$S = \frac{\text{Number of Web services invloved in the composition}}{\text{Total retrieved WSDL links}} \tag{6}$$

It is found that constraint-based clustering is a novel approach and constraint can be changed according to the requirement. Other techniques for integrating the web services includes WS-BPEL which requires separate tool and also it is static version of the composition. But our proposed research is dynamic version of composition method which produces output according to run-time requirement of user.

7 Conclusions

Existing systems are focused on clustering of Web services using machine learning which produces models of context from the terms retrieved from the Web. The training of SVM is done separately for each domain. This system uses the result snippet of the search engines manually such as Wikipedia and Google to get retrieved results. But the proposed system processes the search results of search engine automatically and uses search precision and response time for measure the performance of the system.

The usage of constraint-based clustering in the proposed system not only increases the performance in terms of speed but also saves the storage space. WSDL is big in size which has several elements. Storing all the retrieved WSDL elements consumes more storage space. Retrieving WSDL every time through the internet increases the bandwidth consumption. Hence the proposed system saves the usage of internet in retrieving the remote WSDL and made the system more efficient compared to other systems. Invocation of Web services requires service name element, operation element, input element, target namespace and WSDL link of the suitable Web services. The usage of this technique to store these elements makes the system more efficient because this occupies very less space. It is known that service has different operations and the operation has different input elements which are stored in the cluster. Retrieval of elements of cluster is fast because no need to process the entire WSDL to extract the required elements.

References

1. Sumathi, Pandith, K., Chiplunkar, N., Shetty, S.: Dynamic search and integration of Web services. In: Choudrie, J., Mahalle, P., Perumal, T., Joshi, A. (eds.) IOT with Smart Systems. Smart Innovation, Systems and Technologies, vol. 312, pp. 613–625. Springer, Singapore (2023). https://doi.org/10.1007/978-981-19-3575-6_60
2. Zhang, J., Zhong, F., Yang, Z.: Efficient approach to top-k dominating queries on service selection. In: Proceedings of the 6th IEEE Joint IFIP Wireless and Mobile Networking Conference (WMNC), Dubai, pp. 1–8(2013)
3. R-tree an Article About Data Structure. https://en.wikipedia.org/wiki/R-tree. Accessed 25 Oct 2013
4. Li, Y., Luo, Z., Yin, J.: A location-aware service selection model. Int. J. Serv. Comput. **1**(1), 52–66 (2013). IEEE

5. de Oliveira, R.R., Sanchez, R.V.V.: Comparative evaluation of the maintainability of RESTful and SOAP-WSDL web services. In: Proceedings of 7th IEEE International Symposium on the Maintenance and Evolution of Service-Oriented and Cloud-Based Systems (MESOCA), Eindhoven, Netherlands, pp 40–49 (2013)
6. Ilieva, S., Pavlov, V., Manova, I., Manova, D.: A framework for design-time testing of service-based applications at Bpel level. Serdica J. Comput. **5**, 367–384 (2011)
7. Lu, Z., Jie, W., Patrick, H.: Web services standard-based system resource management middleware model, scheme and test. Int. J. Serv. Comput. (IJSC) **2**(1), 25–44 (2015). ISSN 2330-4472
8. BPMN2 Modeler. https://www.eclipse.org/bpmn2-modeler/. The Eclipse Foundation Copyright© 2016. Accessed 12 Mar 2013
9. Lee, S., Jo, J.-Y., Kim, Y.: Environmental sensor monitoring with secure restful web service. Int. J. Serv. Comput. **2**(3), 30–43 (2014). ISSN 2330-4472
10. Wang, Z., Jing, N., Xu, F., Xu, X.: Cost-effective service network planning for mass customization of services. Int. J. Serv. Comput. **2**(4), 15–27 (2014). ISSN 2330-4472
11. Liu, X., Bouguettaya, A., Yu, Q., Malik, Z.: Efficient change management in long-term composed services. Int. J. Serv. Orientat. Comput. Appl. (SOCA) **5**(2), 87–103 (2010). Springer
12. Nepal, S., Malik, Z., Bouguettaya, A.: Reputation management for composite services in service-oriented systems. Int. J. Web Serv. Res. **8**(2), 1–26 (2011)
13. Joseph Manoj, R., Chandrasekar, A.: A literature review on trust management in web services access control. Int. J. Web Serv. Comput. (IJWSC) **4**(3), 1–18 (2013)
14. Kumara, B.T.G.S., Paik, I., Koswatte, K.R.C., Chen, W.: Improving web service clustering through post filtering to bootstrap the service discovery. Int. J. Serv. Comput. **2**(3), 1–13 (2014). ISSN 2330-4472
15. Sumathi, Chiplunkar, N.N., Ashok Kumar, A.: Dynamic discovery of web services. IJITCS **6**(10), 56–62 (2014). ISSN: 2074-9015. https://doi.org/10.5815/ijitcs.2014.10.08
16. Sumathi, Chiplunkar, N.N.: Necessity of dynamic composition plan for web services. In: Proceedings of 2015 International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT), Davangere, pp. 737–742 (2015)
17. Sumathi, Chiplunkar, N.N.: Populating parameters of web services by automatic composition using search precision and WSDL weight matrix. IJCSE (2018, in press). ISSN:1742-7193