A Survey of Web Service Composition

Aarti Gorade
Rochester Institute Of Technology
Golisano College of Computing
ahg1512@rit.edu

Pooja Shah
Rochester Institute Of Technology
Golisano College of Computing
pks4578@rit.edu

Shailesh Vajpayee
Rochester Institute Of Technology
Golisano College of Computing
srv6224@rit.edu

ABSTRACT

Web services are created and updated dynamically at the run time. Manual web service composition is a convoluted task which demands automation. Numerous approaches have been proposed and implemented to automate composition. Most of them are researched in the areas of AI planning, recommendation, scalability, Quality of Service (QoS) and semantics. This paper summarizes the research work in these areas for automated Web service composition.

Keywords

Web services, composition, QoS, recommendation, REST, semantics, scalability, service repository, natural language processing

1. INTRODUCTION

In recent days, Web services are gaining popularity. Web service is a functionality provided to users to consume resources. It is a modular, autonomous, self-defined system which can be accessed via the World Wide Web. Web service composition utilizes a variety of existing web services without investing time in development cycles. It can efficiently integrate heterogeneous web services on the fly and provide multiple functionalities under the hood. Web service composition provides multiple advantages but it is a complicated and multifaceted problem to compose various web services to satisfy user requirements. Performing all included tasks manually is beyond scope of human abilities. Finding matching service from continuously growing service repository, detection of modifications or updates in existing web service at runtime, combining web services developed on heterogeneous platforms is a hassle and time-consuming task for the end user. To surpass these difficulties, automation of web service composition comes to a rescue. Automation involves automatic generation of a model or discovering of accurate services.

Various researches being conducted to develop or improve the automated web service composition. It falls in diverse domains such as Quality of Service (QoS) of Web Service Composition, Scalability and Service-Oriented Computing, Semantics and Recommendation of Web Services. Quality of Service is one of the most critical requirement of transactions in business process of Web services. Latency and round-trip time are the characteristics of QoS. For web service to perform seamlessly, understanding and analyzing QoS becomes important. QoS for web service composition focuses on various parameters such as availability, performance, reliability, security, etc. Currently, the world invokes services concurrently and scalability comes into picture to be able to handle large number of requests. Service-Oriented Computing is one of the emerging prototype for distributed computing which delivers platform-independent, self-governing, computational components. These components can be published and orchestrated to develop a distributed network with high computational power. Recommendation system provides list of web services to suffice user requirements. Recommender system uses collaborative filtering and content-based filtering approach to recommend a set of web services. UserâĂŹs past behavior is used by collaborative filtering and characteristics of web services are used by content-based filtering to build a predictive model to recommend appropriate services.

Today there is a need of discovering and composing a set of web services from a large pool of already existing web services. Dynamic discovery and composition of Web services can be achieved by performing semantics. Semantics focuses on web services selection based on user inputs which are not bound to any query language and it provides highest simplicity to end-users for web service composition. However, using semantics to understand user intents by performing natural language processing is demanding and interesting. This paper summarizes different research work and various approaches in the above-mentioned domains and mainly focuses on the semantic approach for building automated Web service composition.

The rest of this paper is organized as follows: Section 2 focuses on QoS of the composition. Section 3 illustrates the scalability and service oriented computing. Section 4 describes semantic methods and recommendation system for web services. Section 5 focuses on our implementation plan, challenges in Section 6 and conclusion is drawn in Section 7.

2. QOS OF WEB SERVICE COMPOSITION

Composite services are built using already existing web services where each individual service can have different QoS. QoS of composite service becomes very important and crucial when target application is highly mission-critical [9]. Af-

For class submission.

ter review of research work done in QoS domain, work can be categorized into two main sections such as QoS calculation model for multiple entry and multiple exits (MEME) and QoS-Aware mechanism for web service composition.

2.1 QoS calculation model for MEME

Evaluation of QoS of composite service after considering QoS of each individual component service and ensuring that composite service is capable enough to satisfy requirements of end-users becomes imperative. Evaluation of QoS becomes tricky when one of the selected services has degraded in performance or become unavailable and new component service is required to be chosen at runtime. There are multiple existing methods used for QoS evaluations such as Aggregation method and evaluation of QoS for each execution path. These methods do not consider the probability of each execution path and cannot handle complex patterns such as unstructured conditional or unstructured loop patterns. Zheng et al. [9] proposed new QoS calculation model to overcome those limitations and additionally handled multiple entry and multiple exits (MEME) unstructured loop pattern.

Composite service has main four composition patterns [9]:

- Sequential - Enables next service after completion of previous one
- Parallel - When one service splits into more than two services, which run simultaneously and again merge into a single service
- Conditional - One task splits into exclusive sub-tasks and further merges into another next task
- Loop - Contains arbitrary loop with contains n tasks

For performance evaluation, Zheng et al. used data set of web services with their cost and execution time information [9]. Depending on user requirements there are multiple services that can provide same functionality. Composition plan 1 was created which considered only QoS of web service. Multiple proposed methods to calculate QoS for selecting web services indicates increased performance. Composition plan 2 is generated to satisfy cost and execution time requirement of the QoS paths.

Zheng et al. provided QoS solutions to unstructured loop and conditional patterns [9]. New proposed solution is compared experimentally with already existing methods and shows improvement in performance after considering different execution paths with probabilities incorporating different requirements of users. Comparison is carried out and showed in tabular format for both composition plans. Cost and execution time of each composition plan is compared and it is showed that decline in the few other QoS metrics can help to reduce deviation of QoS of composition. Zheng et al. also provided experimental results to show that imposing standard deviation constraints on the QoS paths of composition service improves the performance.

Other research trends include QoS computation, service modeling and processing. QoS computation considers probabilities of transaction and uses aggression method to aggregate QoS of composition pattern [9]. Service modeling makes use of extended aggregation methods to handle unstructured patterns and uses refined process structure tree (RPST). Aggregation method can recursively aggregate QoS of root that will be final QoS of service composition.

Zheng et al. proposed approach takes care of calculation of QoS information for simple structures along with complex composition structures. Assumptions includes that QoS would be fixed for each task. But in real life, it will be probability distribution. Zheng et al. will be considering probability distribution of QoS as part of future work.

2.2 QoS-Aware mechanism for web service composition

The automation of web service composition does not always guarantee desired QoS. Since many applications which use web service composition are highly scalable, this puts an even higher challenge to match QoS. To ensure QoS there must be a QoS-aware mechanism for web service composition which also supports scalability. Such an approach would enable scalability for dynamic service composition. Chattopadhyay et al. have designed a service composition mechanism which enables a flexible framework to choose between speed and QoS based on application requirements [2].

The problem of a web service composition is when a service query is sent as input, it must be served with a response generated by a composition of web services from the service repository which meets functional dependencies and guarantees QoS [2]. There are two types of web services: sequential and parallel. In sequential web service composition, the output of one service acts as input for another, therefore response time is added sequentially. In parallel composition, services act simultaneously and response time is equal to the service which has highest response time.

The first step is to create a directed acyclic graph which provides the dependencies between the services based on the query [2]. After performing BFS twice on the dependency graph, a subgraph is extracted which has the solution for the input query. Corresponding to the subgraph an auxiliary graph is created with edges representing weights corresponding to QoS parameters. The shortest weighted path, therefore, gives the query solution with the best QoS value.

This approach is tested experimentally on the 2008 Web Service Challenge dataset. The first test is to optimize the number of services in the composition. In this the approximate QoS approach results are deviated from the optimal result which is matched by the A* algorithm, but this is because the QoS approach uses less execution time [2]. There is approximately 25 percent deviation from the optimal result, but the execution times are much much better. The A* algorithm had a service composition time for the complete dataset of around 3.5s whereas the QoS approach took only 5ms.

Using this approach, Chattopadhyay et al. have shown experimentally that we can achieve QoS guarantees without compromising in web service composition time. This approach takes into consideration the QoS of the Web service composition. Also, it has a much better execution time to find the appropriate web service composition as compared to other algorithms like A* and can support customized tradeoff between execution time and QoS. Due to the fast execution time the optimization of number of web services for composition show a 25 percent deviation from the optimal results [2]. This can result in reduced QoS for the web service composition.

3. SCALABILITY AND SERVICE-ORIENTED COMPUTING

Service-oriented computing (SOC) is emerging field that focuses on the transition from data-centric view to service centric view. The Web service technology is exemplary of SOC. After review of research work done in scalability and SOC domain, work can be categorized into three main sections such as Creation of Web service management system (WSMS) for SOC, scalability improvement using graphplan and database approach and middleware for web service mashup.

3.1 Creation of Web Service Management System for SOC

Web service management system (WSMS) provides credible, secure and privacy protection environment. It also caters unified view on the web services management. WSMS focuses on creating composite services where each value-added web service selection happens automatically, queries are modeled to create optimization framework and interactions are secured to preserve sensitive information. Bouguettaya et al. developed a WSMS service oriented digital government system named WebSenior for carrying out daily activities [1]. WebSenior provides services to senior citizens and their families in states with the help of Virginia Department for the Aging (VDA) and Area Agencies on Aging (AAAs). WebSenior works smoothly on heterogeneous platforms, homogeneously blends data and applications and holds autonomy. WebSenior has following important components:

- Service composition ensures whether component services can be combined based on developed rules, composite templates evaluates the soundness of composite services and uses matchmaking algorithm for automatic composition.
- Service optimization executes user-centric optimization based on preferences provided by the user. Score function and aggregation function evaluates composition plans over multiple quality parameters and multiple operations. Moreover, exhaustive and greedy search approaches are used for optimization at the user level.
- Service privacy is preserved using Privacy-Preserving Data Filter (DFilter) and Privacy Profile Manager. DFilter ensures if the user requesting information is authorized or not and PPM enforces privacy at a granular level with the help of privacy profiles stored by individuals.

WebSenior avoids cumbersome paperwork required to do by senior citizens with their AAA and makes the system less complex. It helps AAA to easily access needs of individuals and provide multiple services such as meal delivery, legal service, transportation etc. Semantic web technology, service optimization, service composition technologies are being used in similar other projects [1]. Users evaluated WebSenior application and detailed step-by-step information is provided on how to use WebSenior. Steps include registration, viewing all eligible services, previous orders and their status, granted fund information, choosing a service and placing order successfully. Web service composition and optimization is carried out depending on user's input requirements.

Research trends in this domain include change management of composite services that includes top-down approach and bottom-up approach, web service mining for detecting patterns, and Mobile-Services for mobile users [1]. Change management of web services is divided into two groups - Top down approach and bottom up approach. Owner of the web service invokes top down changes. Web service eventing and Web service notifications are based on events and Bottom Up approach. Web service mining is more flexible than service composition and it is more efficient to find unexpected service combinations that are more efficient. M Services are web services that can be easily accessible by mobile users over wireless network.

WebSenior is a web service management system for senior citizens and it optimizes web service composition dynamically based on user's requirement. Currently, WebSenior evaluation is not performed extensively, but Bouguettaya et al. have planned to do extensive experiments for performance evaluation as part of future work [1].

3.2 Scalability improvement using graphplan and database approach

The previous work made use of in-memory approaches to solve the problem of searching appropriate services from service pool which is a hassle. However, this approach incorporates complex computations which is expensive and search space for it is limited. To mitigate this problem of time and space, skyline operation is applied which helps to improve the scalability and reduce the search space [3]. Li et al. proposed design make use of two approaches: Graph plan and database approach. A planning is constructed by Graph plan. Two stages are used in Graph plan approach: forward expand stage and backward search stage. In forward expand stage, the search graph is constructed and in backward search stage, the solution for web service composition is retrieved. In database approach, Full Solution Indexing using Database (FSIDB) is used to store pre-computed service combinations in the relational databases. In this approach, SQL queries are generated which helps to query the database for the paths which meets the user's requirements. Time to search for an optimal solution for web service composition is improved by storing the popular paths in a separate table by using Partial pre-composing approach.

Other research trends in related work used skyline analysis problem. Pruning of less competitive services and reduction of space overhead is achieved by applying skyline methods. The skyline operator paper used block-nested-loops and divided and conquer algorithms to extend the database by a skyline operation [3]. The skylines computed were then merged by using merging algorithm. Skyline Space Partitioning (SSP) algorithm was used to help provide productive processing of unconstrained skyline queries. Bitmap-based algorithm represented each point into m bit vectors where m determines the number of points. This helped to solve the problems faced by multiple dimensions but has a drawback that it is inapt for dynamic datasets.

Five datasets are generated by using test generator program from WSC-2009. The repository of web services is indicated by each dataset containing WSDL file. Each of the web services included 10 input and around 30-40 output concepts [3]. Performance metrics such as an average time spent to solve the composition problem, optimal response time and optimal throughput were taken into consideration.

They also compared the performance of Graphplan approach and FSIDB approach.

In summary, Graphplan approach returns the optimal solution in a short span of time whereas database approach takes a longer time to search for an optimal solution. Multiple QoS criteria such as bandwidth consumption limit for improved performance is not taken into consideration [3]. Li et al. will be incorporating the multiple QoS criteria as a part of their future work.

In the early days of web services, the composition was done using an application server which forms a bottleneck. End user browser resources solve the problem of the bottleneck [6]. But this method increased the need to address heterogeneity in the languages used and communication coordination. Wohlstadter et al. have described a middleware which performs selective partitioning of XML pipelines between browser client and application server [6]. A pipeline consists of processes interconnected with pipes from the output of one process to the input of other. The components are executed through partitioning configuration.

3.3 Middleware for web service mashup

The architecture component favs determine query string sent to an eBay Web Service, xslt stylesheet transforms eBay response. There are four different approaches for partitioning of components [6]. In AS approach, an eBay and xslt components run on the server side but have scalability issues. To improve the scalability, EC or EXC approaches can be used. EXC requires xslt support however, EC does not require it. Wohlstadter et al. proposed and implemented EC approach for partitioning of XML pipelines to support the heterogeneity of different clients and reduced performance overhead.

Manual web service composition of the static application has shown promise, but to automate this process an architectural framework is required. Paik et al propose a four-stage architecture for Automatic Service Composition (ASC) [5]. The basic four stages are workflow planning, finding services in a registry, selection from the shortlisted services and their consequent execution. Such a stage process is useful because it is straightforward and enables exception handling. This architecture needs an orchestration manager for nested workflow and a transformer for composition property which will reduce the complexity of the ASC. The architecture shows that for small tasks service selection cost is less than that of workflow planning, but service selection cost is inversely proportional to the number of nested levels while the cost of workflow planning remains constant.

Evaluation of the approach is performed in five parts. In the first part the computational cost of service composition is tested. With increase in depth of nesting the computational cost increases exponentially [5]. Since selection for service composition takes 76 percent of the total time and with increase in depth the selection time increases. A similar result is obtained if computational cost is tested with the number of tasks though selection time is 36 percent. Cost of workflow generation is faster than service selection when number of tasks are small. In the third part computation cost is compared with number of constraints. Increase in number of constraints results in linear increase in computation cost. Then the test is performed for computation cost with number of service instances per task. This resulted in an exponential growth with increase in number of service instances. In the last test, computational cost is measured with respect to number of classes in domain. This test shows that it is independent of classes [5].

All approaches related to web service composition have four categories. The first category is related to workflow planning, in which AI strategies are used. The second category is related to planning and discovery of web services for composition, in which planning on heterogeneous ontology is used. The third is discovery and selection for execution of web services. The fourth is selection for whole composition in which there could be QoS constraints. The approach of Paik et al. focuses on an orchestration of nested workflow in service composition which provides functional scalability and framework [5].

The proposed architecture is highly scalable, seamless and adaptive. However, service selection computational cost increases exponentially with increase in number of tasks in workflow [5].

4. SEMANTICS AND RECOMMENDATION OF WEB SERVICES

Nowadays, Web service recommendation domain is growing very fast and has gained high importance. Choosing correct web service from easily available abundant web services has become critical. After review of research work done in semantics and recommendation of web services domain, work can be categorized into three main sections such as time-aware service recommendation framework, composition of non-Web service components using SAWSDL, RESTful web services.

4.1 Time-Aware service recommendation framework

Content matching, prediction QoS, semantic matching and hybrid approaches are the few crucial techniques used for recommendation of web services. Previous work in this domain did not check for most evolved version of service and did not consider functional requirements of each mashup. Joint analysis of temporal information, topology, and content, performed by Zhong et al. removed limitations. Latent Dirichlet Allocation (LDA) is used to extract topic activity timing information and service correlation matrix giving information about usage history. This helped to outlook evolution of topic and prediction of activities of service. Zhong et al. proposed time aware service recommendation framework [10]. This framework uses mashup description based collaborative filtering with predictive service activities and service description based content matching as a platform. This framework consists of three components:

- Temporal information (TI) extraction: Uses service usage history to predict service activity
- Mashup-description-based collaborative filtering(MD-CF): Recommends based on historical mashups having same functional requirements
- Service-description-based content matching (SDCM): Recommendation is performed based on semantic similarity between functional requirements of mashup and content description of service

Real world data analysis and experiments showed improvement in performance due to consideration of temporal information. For evaluating proposed model, the dataset is prepared from ProgrammableWeb.com using metadata information of each service and mashup. Preprocessing included generation of original words, pruning, stripping suffix and spell correction [10]. Dataset is divided into training and testing dataset using timestamp as cutoff. Mean Average precision is an evaluation metric used. Time aware service is compared with other generated combination methods MDCF, SDCM and TI.

For information retrieval and recommendation system, Mean Average Precision (MAP) is used [10]. A Higher value of MAP indicates high accuracy. Proposed model is compared with individual recommendation systems such as MDCF, SDCM or TI or combinations of them. A step forward, experimental results show that MDCF, SDCM, TI together perform excellently than other combinations. Content, topology and temporal information together provide better results.

4.2 Composition of non-Web service components using SAWSDL

Nowadays, the composition of non-Web service components with Web service-based components is gaining importance. Searching for compatible and functionally equivalent components from a large set of available of non-Web service-based components requires tremendous knowledge of programming languages [4]. To alleviate this problem, Sheng et al. proposed progressive composition framework where technologies related to semantic Web and Web services matching are applied. Various technologies such as Semantic Annotation for Web Service Description Language (SAWSDL), searching and matching algorithms are applied. SAWSDL makes use of semantic annotation and WSDL to mash up various components. However, the components should be annotated beforehand while using SAWSDL. Failing to annotate the components will not mash up the components. Searching and matching algorithms calculate the score of the compatibility of the component which helps to rank the potential components[4].

Various WSDL files were downloaded from public domain Web service portals: XMethods Web Services Portal and The QWS Data-Set [4]. The effectiveness of the application is tested with increasing number of WSDL files. It has been observed the number of WSDL is exponentially proportional to the time it takes for completing the match.

Various research trends compared the mashup tools and approaches based on the types of components, methods for the users to find suitable components and alternative mashup programming patterns to combine the various services [4]. Instead of only URL-based sources in Yahoo! Pipes, DAMIA extended the type of data sources for mashup. These data sources include Excel, Notes, Web Services and XML-document [4]. Yahoo! Pipes doesn't provide features for accepting the arbitrary inputs or outputs, flow of the pipe is also static and sequential. Since Yahoo! Pipes is a server based technology, user has no freedom to construct and execute the pipe using locally stored data. With the help of Damia application, new mashup operators could be created making it more powerful than Yahoo! Pipes operators [4].

Based on methods for the users to find suitable components, various mashup platforms do not meet the user requirements to help them discover the suitable services. Mashup Automation with Runtime Orchestration and Invocation (MARIO) [4] helps the user to find the suitable services by recommending the services to the users. Exploration of available mashups and preview of composition results is performed using tag-based service description, service selection and taxanomy.

Based on alternative mashup programming patterns, Karma and UQBE created an environment where various data sources are integrated by using common attributes thereby making it easier for the users to understand the data semantics [4]. In recent days, spreadsheet programming paradigms for mashups are recommended. In Mashroom, Web-extracted data can be viewed using nested relational model.

The proposed solution doesn't require the users to have a low-level knowledge during the composition of the non-Web service components with Web-service based components [4]. In real life, it is crucial to allow components to dynamically expose their capabilities for mashup which was not taken into consideration. Sheng et al. plan to allow the component to dynamically expose their capabilities for mashup as a part of future work. Future work also largely focuses on combination of service mashup approach with a process-based integration approach.

4.3 RESTful Web services

There are two types of Web services: SOAP and RESTful Web services. SOAP web services allow only XML format data. RESTful web services support JSON, XML, etc. data formats. Using RESTful web services has several advantages [7]:

- They are light weight since they use HTTP
- Easily accessible as URIs (Universal Resource Identifiers) can be shared
- They support caching and partitioning of URI which makes them scalable
- They are declarative since they explain the resources

The end-user needs to perform several tasks to find the services that meet their criteria and needs. Searching for the best services that meet their goals and constraints is a hassle and online tasks are unable to automatically analyze the end-users goals. The previous work to solve this problem focused on extracting relevant tasks from online How-to instructions. The previous work didn't take into consideration the user constraints. To surpass these limitations, Zhao et al. proposed a framework which uses natural language processing techniques to identify the user goals and recommends the best services to the user that meet their criteria and needs [8].

Since RESTful Web services are stateless, the client may need to send redundant information which increases network traffic. It is difficult to represent resources in URI if there is no simple hierarchy in the storage method of the web resource.

5. OVERVIEW OF IMPLEMENTED FRAMEWORK

We have selected semantic analysis approach using NLP to perform web service selection and composition as mentioned in [8]. It provides an efficient framework to achieve automation within the user constraints, which makes it an appropriate candidate for implementation. The framework contains multiple sub-components which help in selection and composition of web services that meet the user goals. We used multiple tools such as Stanford Part-of-Speech (POS) Tagger and Stanford Natural Language Parser , JSoup package and Lucene. Each word in a sentence is tagged by POS Tagger with part-of-speech marker. Grammatical structure of a sentence is analyzed by Stanford Natural Language. It groups the words into phrases and generates a tree with grouping of words. JSoup package is a Java HTML parser which parses HTML page and can extract the text from the HTML page. This is used to generate the final query statement term vector which is used to search the index of the database to get matched APIs.

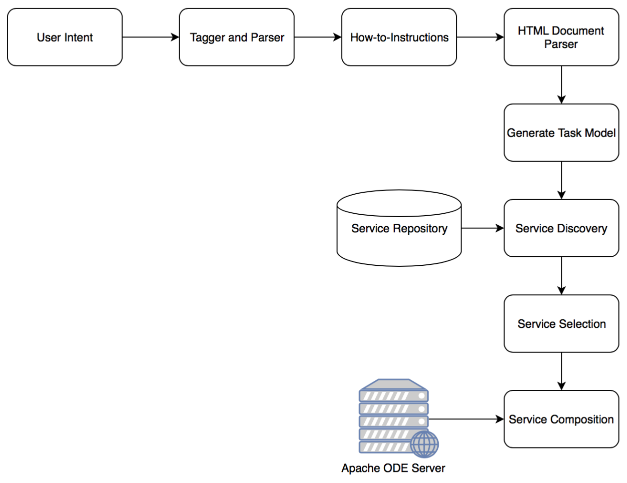


Figure 1: Implementation Framework

Figure 1 shows pictorial representation of the implementation workflow. Below are the implemented components along with their functionality:

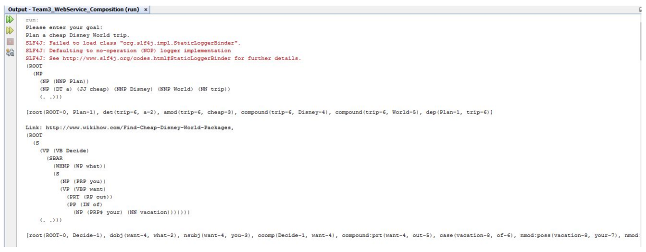
- **User input**: Take goal as an input from the user. This will be a regular text format. Our implementation is focused only on English language. For example: Plan a cheap Disney World trip.

- **Tagger and Parser**: Stanford Part-of-Speech (POS) Tagger is used to identify the nouns, prepositions, and verbs from the user input goal. Stanford Natural Language Parser uses the output of POS Tagger as an input and provides a list of noun phrases. These noun phrases are concatenated and passed to How-to-Instructions component to extract the relevant URL of eHow or WikiHow website. Figure 2 shows the sample output for Tagger and Parser for user input Plan a cheap Disney World trip.

- **How-to-Instructions**: Google Custom Search engine is configured to provide the URLs only from WikiHow and eHow websites. WikiHow and eHow websites are knowledge bases for How-to instructions. These how-to instructions provide users with step-by-step information to perform a particular task. Google Custom Search API is invoked with noun phrases as a query parameter. The response of Google custom search API includes a set of relevant URLs from How-to instruction knowledge bases. This response contains the URLs according to their ranking score. We selected first URL which is returned in a response as Google uses page-ranking algorithm to give the most relevant pages that matches the query. Figure 2 shows the relevant link fetched from eHow or WikiHow website that matches the user's goal.

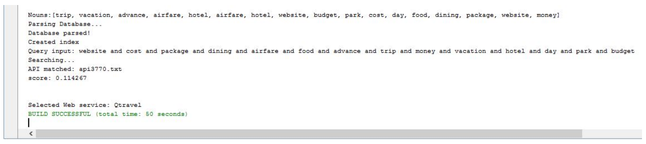
- **HTML Document Parser**: There is a standard format followed by WikiHow and eHow knowledge bases. The format contains multiple steps as a heading and detailed description about that particular step. We developed a code using JSoup package in Java to parse the HTML page. We extracted the the how-to instructions steps from the HTML document.

- **Generate a Task Model**: Each step is parsed again with Stanford Part-of-Speech (POS) Tagger and Stanford Natural Language Parser to get nouns. The set of nouns act as a input to the Lucene, a text search engine, for discovering services.



```
run:
Please enter your goal:
Plan a cheap Disney World trip.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
(ROOT
  (NP
    (NP (NNP Plan))
    (NP (DT a) (JJ cheap) (NNP Disney) (NNP World) (NN trip))
    (. .)))

[root(ROOT-0, Plan-1), det(trip-6, a-2), amod(trip-6, cheap-3), compound(trip-6, Disney-4), compound(trip-6, World-5), dep(Plan-1, trip-6)]

Link: http://www.wikihow.com/Find-Cheap-Disney-World-Packages,
(ROOT
  (S
    (VP (VB Decide)
      (SBAR
        (WHNP (WP what))
        (S
          (NP (PRP you))
          (VP (VBP want)
            (PRT (RP out))
            (PP (IN of)
              (NP (PRP$ your) (NN vacation)))))))
    (. .)))

[root(ROOT-0, Decide-1), dobj(want-4, what-2), nsubj(want-4, you-3), ccomp(Decide-1, want-4), compound:prt(want-4, out-5), case(vacation-8, of-6), nmod:poss(vacation-8, your-7), nmod
```

Figure 2: Output of Tagger and Parser, How-to Instructions



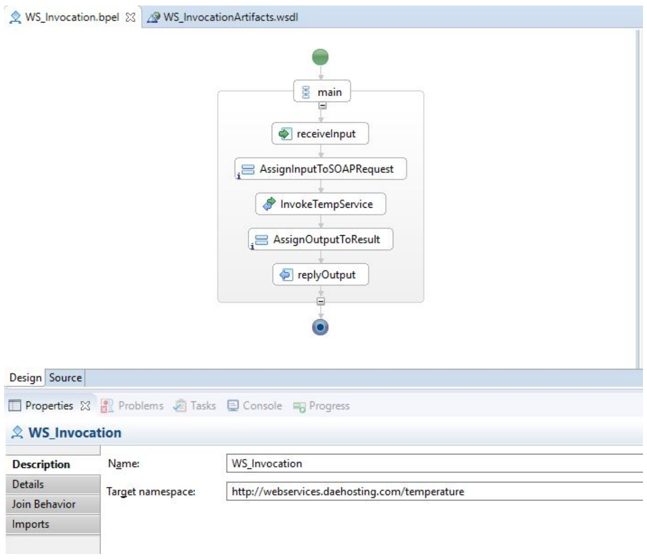
```
Nouns:[trip, vacation, advance, airfare, hotel, airfare, hotel, website, budget, park, cost, day, food, dining, package, website, money]
Parsing Database...
Database parsed!
Created index
Query input: website and cost and package and dining and airfare and food and advance and trip and money and vacation and hotel and day and park and budget
Searching...
API matched: api3770.txt
score: 0.114267


Selected Web service: Qtravel
BUILD SUCCESSFUL (total time: 50 seconds)
```

Figure 3: Output of Service Discovery

- **Web Service Discovery**: Lucene maintains an index for the ProgrammableWeb database of APIs. The ProgrammableWeb database (represented as a text file) is parsed according to its attributes and every record is saved as a separate text file. This makes retrieval of similar content quicker than database access. The term vector generated in the previous steps is used as input to query the index. Lucene uses Vector Space Model and Boolean Model to perform search operations and calculate the relevancy score for each service. The score of each service is calculated by comparing the nouns which were extracted with the service description retrieved from the service repository. Figure 3 shows the output of API discovery, API score and selected Web service API.

- **Web Service Selection**: To select the appropriate service selection we make use of the categories provided for the APIs included as an attribute in the database. The user is provided top 10 APIs from top 10 relevant categories.

- **Web Service Composition**: Composition of web services is done using BPEL(Business Process Execution Language) which is used to define and automate execution of a business process. BPEL allows top-down approach for SOA and enables composition, orchestration and coordination of services. With BPEL multiple web individual web services can be combined and a new composite service is generated. There are multiple tools available for designing BPEL for a business process such as Eclipse BPEL designer, RedHat JBoss developer studio, Camunda modeler, etc. We have used Eclipse BPEL Designer on Eclipse IDE to automate composition. BPEL requires SOAP based services therefore, WSDL file of the web services are required for linking the web services. Once the services are composed a server is required to host the service. Apache ODE server to host the service and testing is done using Web Services Explorer. Figure.4 and Figure.5 show graphical flow of composition of web services in BPEL Designer.



Name: WS_Invocation
Target namespace: http://webservices.daehosting.com/temperature

Figure 4: BPEL work-flow

5.1 Algorithm

1. Take user goal as an input
2. Give the user goal as an input to Stanford POS Tagger
3. Identify the nouns, prepositions and verbs
4. Give the output from step 3 to Stanford Natural Language Parser
5. Extract the noun phrases
6. Pass the noun phrases as a query parameter to Google Custom Search API
7. Get the relevant URL of eHow or WikiHow websites

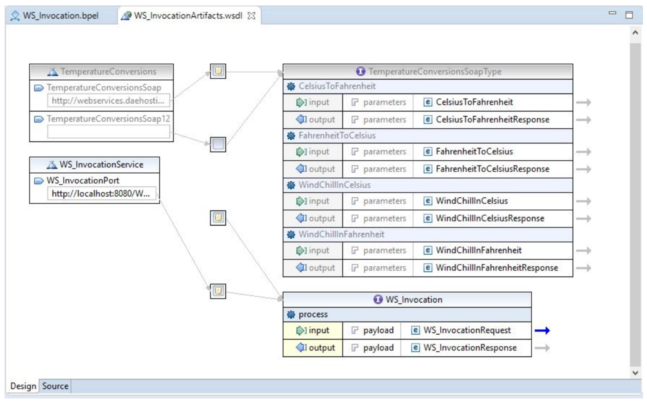


Figure 5: BPEL design

8. Parse the HTML page and extract all the steps along with its detailed description
9. For each step:
   (a) Tag using Stanford POS Tagger
   (b) Parse using Stanford Natural Language Parser
   (c) Extract nouns
   (d) Add the nouns to final set of nouns
10. Use final set of nouns retrieved from step 9(d) and give it as an input to Lucene engine
11. Run the Lucene engine for each API description retrieved from ProgrammableWeb database to calculate relevancy score
12. Discover the services based on relevancy score
13. Select top 10 APIs from the repository
14. Develop web service composition for user selected APIs using BPEL.

Table 1: Query Examples: Total-Num: Number of selected APIs (max 10 out of 11163); User-Selected-Num : APIs used by user (out of 10) ; Actual-Used: Total APIs actually used by user to achieve goal;

Query	Total-Num	User-Selected-Num	Actual-Used
I would like to get a server which will allow me remote access	10	3	3
book a flight for travel	10	4	5
find me an online hotel booking service	9	6	6
shopping service	10	4	6
Plan a trip to London	10	4	4
Burn fat and stay healthy	8	2	3
need to book a hotel	10	4	4

6. EVALUATION

Our implementation provides a list of 10 APIs from 10 different categories which have the most similarity to the parsed steps from the HTML files. We asked 10 Computer Science Graduate students to run our application and grade it on a scale of 10. We received an 82 percent satisfaction from the users. 4 users used APIs from the list to achieve their goal. 5 users used some APIs from the list and some from their own research. Only 1 user did not find any API useful from the list provided. Since the search of the APIs is dependent on how the APIs are described in the repository, it can sometimes happen that relevant API matches are not found for an input query.

The results in Table.1 show that our application is indeed efficient at helping users with the selection of services. Also, our application is easy to use since it does not require any specific query language for input.

The results in Table.2 show that the users indeed found our application to be useful, simple, efficient and time saving.

Table 2: Questionnaire

Summary	Questions	Score (out of 10)
Usefulness	Is the application helpful to select services?	8
Simplicity	Is it easy to use the application?	9
Accuracy	Did the application provide APIs which you actually used?	7
Time Saving	Did the service save your time to find the services?	9

7. CONCLUSION

[h] We have summarized research papers focused on web service composition with optimal QoS, improving scalability, semantic analysis for recommendation of relevant web services based on users' requirements. Based on this review we selected a semantic analysis approach and used Natural Language Processing (NLP) to understand the user intents. We performed service discovery and selection successfully using Lucene engine. Our future work focuses on selection of most appropriate web service to satisfy a particular users' requirements and to generate a composite service by combining multiple services and also ensure that developed web service composition is most efficient and ease of use to the end-user.

8. REFERENCES

[1] A. Bouguettaya, Q. Yu, X. Liu, and Z. Malik. Service-centric framework for a digital government application. *IEEE Transactions on Services Computing*, 4(1):3–16, Jan 2011.

[2] S. Chattopadhyay, A. Banerjee, and N. Banerjee. A scalable and approximate mechanism for web service composition. In *2015 IEEE International Conference on Web Services*, pages 9–16, June 2015.

[3] J. Li, Y. Yan, and D. Lemire. Scaling up web service composition with the skyline operator. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 147–154, June 2016.

[4] A. H. H. Ngu, M. P. Carlson, Q. Z. Sheng, and H. y. Paik. Semantic-based mashup of composite applications. *IEEE Transactions on Services Computing*, 3(1):2–15, Jan 2010.

[5] I. Paik, W. Chen, and M. N. Huhns. A scalable architecture for automatic service composition. *IEEE Transactions on Services Computing*, 7(1):82–95, Jan 2014.

[6] E. Wohlstadter, P. Li, and B. Cannon. Web service mashup middleware with partitioning of xml pipelines. In *Proceedings of the 2009 IEEE International Conference on Web Services*, ICWS '09, pages 91–98, Washington, DC, USA, 2009. IEEE Computer Society.

[7] H. Zhao and P. Doshi. Towards automated restful web service composition. In *2009 IEEE International Conference on Web Services*, pages 189–196, July 2009.

[8] Y. Zhao, S. Wang, Y. Zou, J. Ng, and T. Ng. Mining user intents to compose services for end-users. In *2016 IEEE International Conference on Web Services (ICWS)*, pages 348–355, June 2016.

[9] H. Zheng, W. Zhao, J. Yang, and A. Bouguettaya. Qos analysis for web service compositions with complex structures. *IEEE Transactions on Services Computing*, 6(3):373–386, July 2013.

[10] Y. Zhong, Y. Fan, K. Huang, W. Tan, and J. Zhang. Time-aware service recommendation for mashup creation in an evolving service ecosystem. In *2014 IEEE International Conference on Web Services*, pages 25–32, June 2014.