International Journal of Grid and Distributed Computing
Vol. 9, No. 8, (2016), pp.181-196
http://dx.doi.org/10.14257/ijgdc.2016.9.8.16

A Semantic-Aware Approach for Automatic Cloud Services Composition

[1*] Hasan A. H. Naji, Chao Zhong Wu[2] and Shu Gao[3]

[1,2]Intelligent Transport Systems Research Center, Wuhan University of Technology, 1040 Heping Avenue, Wuchang District, Wuhan, 430063, P.R. China.*
[3]School of Computer Science and Technology, Wuhan University of Technology, 1040 Heping Avenue, Wuchang District, Wuhan, 430063, P.R. China.
[1]E-mail: hasanye1985@gmail.com, [2]wucz@whut.edu.cn, [3]gshu418@163.com*

Abstract

Service composition is considered as a promising approach to generate composite services for satisfying services Consumers' needs using their preferences. Literature review shows that previous studies ignored issues that affect Service composition such as increasing composition time and no ranking for produced composite services. To address these issues, we propose a semantic-aware approach for automatic cloud services composition. Firstly, a semantic network of service is built using an efficient input-output matching algorithm. Secondly, a Forward-Chaining based algorithm is adopted to create composite services, according to the request of the Cloud Service Consumer. Thirdly, a hybrid ranking method based on similarity quality and QoS Criteria is implemented to select "the most suitable" composite service. The results of experiments provide evidence that our approach can effectively produce composite services for service consumer's request, and show benefits of consuming less composition time comparing to similar composition algorithms.

Keywords*: Semantics, Cloud Service, Composition, Semantic Network, Forward-Chaining, QoS*

1. Introduction

Services Composition concentrates on defining and designing models and methods for performing composition on existing services to generate a more complex service, which satisfies the required functionalities of services consumers.

The composition work can be categorized into several types based on the approaches and technique used [1]; namely Static and Dynamic Composition, Model Driven Service Composition, Declarative Service Composition, Semantic Based Composition and Context Based Web Service Discovery and Composition.

For supporting automatic concepts in Services Composition, Semantic Annotation is presented. Ontology concepts are adopted to provide semantics to services. Semantic Services Composition [2] is a model that improves the flexibility of services in an application. An automatic services composer should integrate the required services, which are included in a composition plan, according to the user's specifications. The ability of automatically composing services and generating new composite services is main points for Semantic Services Composition. Furthermore, composite services [3] are dynamic, i.e., their components are able to be automatically selected at run-time.

There is a number of studies and researches conducted in the field of semantic and automatic cloud services composition. In [4], authors presented a framework called DynamiCoS for performing automatic services composition. Cloud services with semantics are used in DynamiCoS to support semantic reasoning which in turn facilitates automatic service discovery, selection and composition. Serrano Martín et al. [5]

ISSN: 2005-4262 IJGDC

introduced a semantic-based approach of services provision for Cloud Service Composition. Domain ontological modelling and semantic links method are adopted for supporting services composition. Tan W et al. [6] proposed an approach for services composition. In the proposed approach, the inputs-output parameters of the service operations are modeled as colored places, and service operations themselves are represented as transitions along with input/output places. Composition plans are generated by linking services using input-output similarity matching. Peter Bartalos and Ma´ria Bielikova [7] defined a QoS and behavior-aware approach for cloud service composition that deals with changes in the services and continuing arrival of new comings of composition queries. Omer A M and Schill A[8] introduced an input/output dependency based approach for automatic composition plans creation. In this approach, Input-Output dependency is described as directed graph and composition plans are created based on a graph traversal method called the topological sorting.

Most of the composition works ignored issues affect Service composition such as increasing composition time and no ranking for generated composite services. The main contribution of this paper is proposing and presenting in detail a semantic-aware approach for automatic cloud services composition that can explicitly consider the above issues. This approach can automatically find the most suitable composite service for satisfying a given request, which includes requested inputs, requested outputs and QoS weights, in a possible execution time. In addition, the approach provides a ranking method for the case wherein many composite services may produce from the composition process.

The rest of the paper is organized as follows: section 2 presents a Literature Review related to the study topic. Section 3 introduces the proposed approach in detail. Section 4 introduces experiments and analysis of our approach. In section 5, we summarize the content and the findings of this study.

2. Literature Review

2.1 Cloud Services

Fast development in the using of Cloud Computing technology leads to deploying more and more Cloud Services to cloud services pool. The services provided by Cloud providers are divided into following three main categories. Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS). Software as a Service (SaaS)[9]. This cloud service presents an application as a service without need to install and run any application on the computer of the cloud consumer. Consumers of SaaS Services, who are usually end users of applications or software administrators, can access this type of software using web browsers or mobile apps. Platform as a Service (PaaS). This type provides a developing platform as a service. In common, a PaaS service provides a complete development platform for organizations requiring a development instance of an application. Developers, testers, deplorers and application administrators are PaaS service consumers. Google App Engine is an example of a popular PaaS. Infrastructure as a Service (IaaS). This type of services provides infrastructure as a service, including server CPU cycles, data center space, storage resources, and database capacity. Services in IaaS are billed on a per use basis, the capacity can be increased in small increments, and the service is governed by stringent SLAs [10]. Typical IaaS consumers consist of system developers, network engineers, system administrators, monitoring engineers and IT managers.

This paper focuses on software as a service (SaaS) and presents study to discuss Cloud Service Composition.

2.2 Unified Ontology

In this study, a unified ontology, which collects knowledge about cloud services, including Services Domains, Cloud service Characteristics and QoS Criteria, is built to unify the description of Cloud Services as described in [11]. This ontology defines a unified model that serves as a semantic-based repository, and plays a vital role in Cloud Service Advertisement, Selection and Composition. The Unified Ontology, which adopts OWL-S language [12], consists of classes and properties. The classes present the main terms and concepts of services, while properties present the relationships between the classes. The Unified Ontology contains the following classes as shown in **Figure 1**.

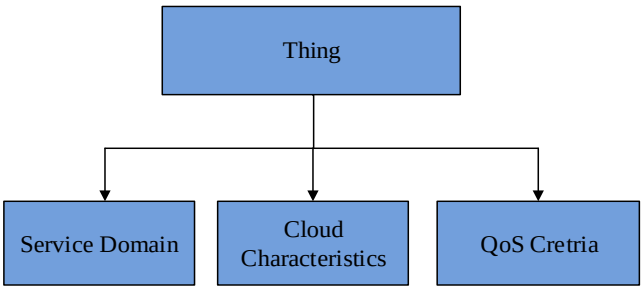

Figure 1. Snapshot of Main Concepts in Unified Ontology

The Unified Ontology contains main classes, including Service Domain, Cloud Characteristics and QoS Criteria.

2.2.1 Service Domain

A Service Domain presents classes for describing the functions of services according to their domain. Due to the complexity of covering and representing the whole domains of services, in this section we introduce the Finance domain as an example of Cloud Services domain.

Finance Domain provides semantic classes for Cloud Service descriptions in Finance area. A Finance Domain ontology is built to assist in Cloud Service advertisements. **Figure 2** shows the main classes in the Finance Domain which has several key concepts including *Currency*, *financial_Instrument*, *finance_operation*, *order, agent*, *quality* and *finacial_mean*.

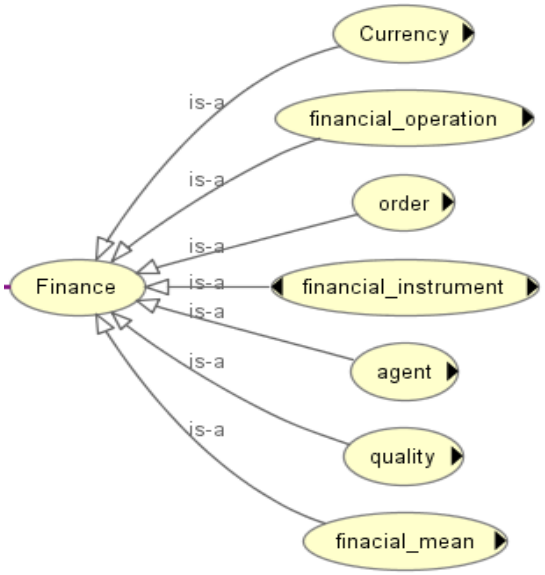

Figure 2. Snapshot of Main Classes in Finance Domain Ontology

Each class represents its area using a subclass, for instance, Currency_*Service* and *Conversion_Type* are subclasses of *Currency* class.

2.2.2 Cloud Service Characteristics

Cloud Service Characteristics [13] are specific types of characteristics a Cloud Service provides to Cloud Services' Consumers. There are many main Cloud Service characteristic included in the Unified Ontology. The value partitions pattern is adopted in order to restrict the range of possible values for each characteristic into an exhaustive list, e.g. the range of possible values for the *PaymentType* characteristic is Free, Dynamic or PayOnDemand. A part of the ontology of Cloud service characteristics is presented as shown in **Figure 3**. There are several key classes in Cloud Service Characteristics' Ontology, including *CloudServiceOpenness*, *PaymentType*, *LicenceClass*, *IntendedUserGroup* and *ServiceAgreement*.

The main difference between the traditional services and Cloud Services is the latter are billed based on dynamic use. Rather than paying a fixed monthly or yearly charge, the customer only pays for the resources he uses on the consuming time.

1) QoS Criteria

QoS Criteria [14] of a Cloud Service are represented as concepts in the Unified Ontology. The QoS Criteria included in the QoS criteria ontology are *Price*, *ResponseTime*, *Reputation*, *Reliability* and *Availability*. The value of each *QoS criteria* guaranteed by the Cloud Service Providers are described using data properties, including *hasResponseTime*, *hasAvailability*, *hasPrice*, *hasReputation* and *hasReliability*. Data properties range can be one of the supported data types e.g. integer, string, boolean, etc. For example, in order to specify the price of the most popular plan of the Cloud Service *Salesforce* is 90. In the unified ontology we use the following data property assertion: *hasPrice* 90 on the Salesforce service.

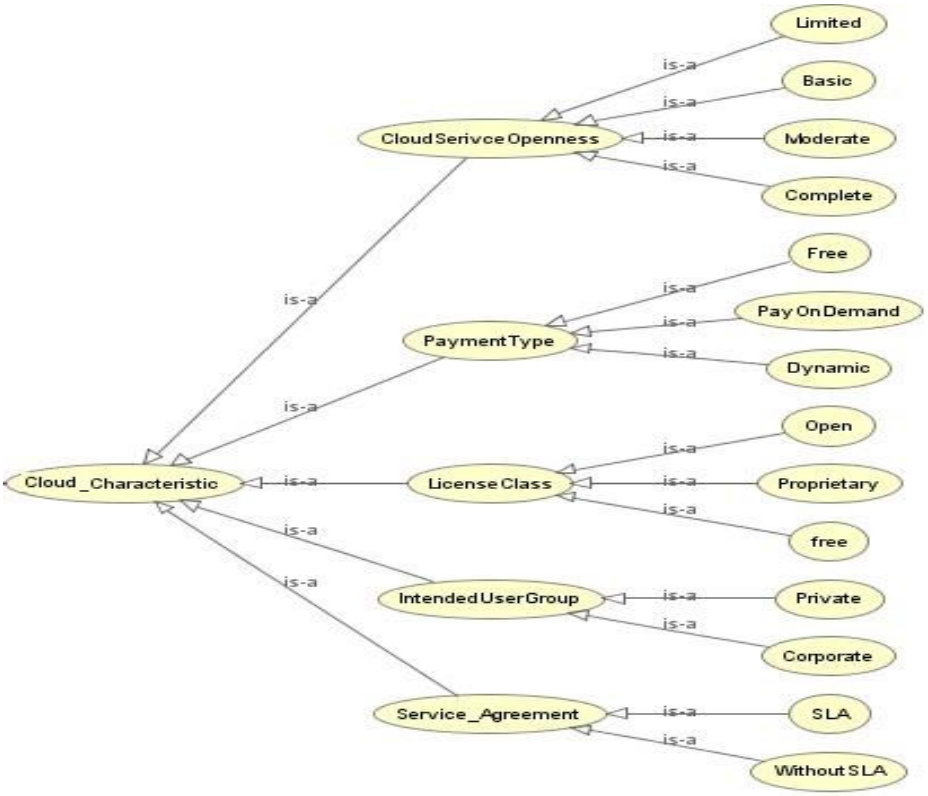

Figure 3. Snapshot of Main Concepts of Cloud Service Characteristics Ontology

3. Semantic-Aware Approach for Automatic Cloud Services Composition

In this section, a Semantic-aware approach for automatic Cloud Services Composition is introduced. The Semantic-aware approach contains several novel algorithms including Output-Input matching algorithm for building a semantic network, Forward-Chaining algorithm for generating composite services and hybrid ranking method for finding the optimal composite service.

3.1 Efficient Output-Input Matching Algorithm for Building a Semantic Network

As the description of cloud services (i.e inputs-outputs, QoS Criteria and Cloud service characteristics) are stored in the Unified Ontology, the concept of Semantic Network [15] is adopted to facilitate services composition and to reduce consuming time for composition. A Semantic Network Sem_{net} is a directed graph of cloud services (in a specific domain) where these services are linked by finding the matching similarity between their inputs and outputs. The matching similarity is performed using Semantic Casual Links [16]. Semantic Causal Link is a link of a semantic relationship between inputs and outputs of two cloud services.

The outputs and inputs of cloud services are mapped to concepts in the Unified Ontology. Therefore, retrieving the semantic relationship between two cloud services s_1 and s_2 is similar to discovering the semantic similarity between two semantic concepts ($Out\text{-}s_1$ and $In\text{-}s_2$), where $Out\text{-}s_1$ is the concept of the output of cloud service s_1 whereas $In\text{-}s_2$ is the concept of the output of cloud service s_2.

The function $Sim(Out\text{-}s_1, In\text{-}s_2)$, called similarity function, is used to find semantic relationship between two cloud services s_1 and s_2. In other words, $Sim(Out\text{-}s_1, In\text{-}s_2)$ function determines the semantic matching type between two concepts ($Out\text{-}s_1$ and $In\text{-}s_2$). The matching types [17] may obtained from $Sim(Out\text{-}s_1, In\text{-}s_2)$ are: (i) Exact ($\equiv$): In this type, the output concept $Out\text{-}s_1$ of a cloud service s_1 and the input concept $In\text{-}s_2$ of a cloud service s_2 are semantically equivalent; formally: $Out\text{-}s_1 \equiv In\text{-}s_2$. Exact type takes the value 1. (ii) Plug-In ($\hat{o}$): In plug-in type, the output concept $Out\text{-}s_1$ of a cloud service s_1 is a sub-concept of the input concept $In\text{-}s_2$ of a cloud service s_2; formally: $Out\text{-}s_1 \hat{o} In\text{-}s_2$. Plug-in type takes the value 1/2. (iii) Subsumed ($\tilde{o}$): Subsumed type indicates that the output concept $Out\text{-}s_1$ of a cloud service s_1 is a super-concept of the input $In\text{-}s_2$ of a cloud service s_2; formally: $Out\text{-}s_1 \tilde{o} In\text{-}s_2$. Subsumed type takes the value 1/3. (iv) Disjointed ($\perp$): In this type, the output concept $Out\text{-}s_1$ of a cloud service s_1 and the input $In\text{-}s_2$ of a cloud service s_2 are none of the matching types mentioned above; formally: $Out\text{-}s_1 \perp In\text{-}s_2$. Disjointed type takes the value 0. For simplicity, the function $Sim(Out\text{-}s_1, In\text{-}s_2)$ feeds back the corresponding value of matching type, such as 1 for Exact type.

As in [16], a Semantic Causal Link involves the semantic relationship between input and output of two cloud services. A Semantic Causal Link is defined as a triple$< s_1, Sim(Out\text{-}s_1, In\text{-}s_2), s_2 >$. s_1 and s_2 refer to two cloud services in a specific domain. $Out\text{-}s_1$ is a concept for an output of the cloud service s_1 whereas $In\text{-}s_2$ is a concept for an input of the cloud service s_2. The function $Sim(Out\text{-}s_1, In\text{-}s_2)$ feeds back the matching type (the corresponding value) according to the matching degree between the concepts $Out\text{-}s_1$, $In\text{-}s_2$ in the Unified Ontology. A Semantic Causal Link $< s_1, Sim(Out\text{-}s_1, In\text{-}s_2), s_2 >$ represents that: (i) s_1 precedes s_2, since an output of s_1 is consumed by an input of s_2 and (ii) No intermediate cloud service is existed between s_1 and s_2.

A Valid Causal Link is a Semantic Causal Link $< s_1, Sim(Out\text{-}s_1, In\text{-}s_2), s_2 >$ when the value of $Sim(Out\text{-}s_1, In\text{-}s_2)$ is > 0. Finally, a Semantic Network Sem_{net} is a set of cloud services (in a specific domain) where these services are linked by Valid Casual Links.

For instance: we have a set of cloud services in the domain of weather. Each cloud service is represented as the tuple $s_i =<$id, name, ins, outs$>$ where id indicates a cloud service identifier, name denotes a cloud service name, ins is a set of input(s) of the cloud

service s_i and outs is a set the output(s) of the cloud service s_i. For the sake of simplicity, we assume that each cloud service has one input and one output. Table 1 shows of cloud services and their descriptions.

Sim(*Out-s_1, In-s_2*) function is used to determine the semantic matching type between inputs and outputs in **Table 1**. **Figure 4** shows a part of the semantic network, which is built according to the descriptions of cloud services in **Table 1**.

Table 1. Cloud Services' Descriptions

	Id	Name	Input	Output
s_1	1	FindZipCode	City	Zipcode
s_2	2	FindLatLong	Zipcode	Lat/long
s_3	3	GetPressure	Lat/long	Press
s_4	4	GetWeather	Lat/long	weather
s_5	5	Findweather	City	weather
s_6	6	Findclothesforweather	Weather	clothes

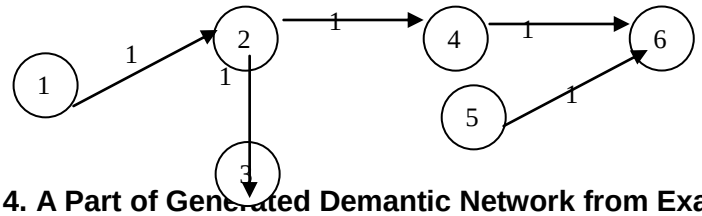

Figure 4. A Part of Generated Demantic Network from Examples in Table 1

The output of the service s_1 is similar (exact) to the input of the service s_2; and the output of the service s_2 is similar (exact) to the input of the services s_3 and s_4. **Algorithm 1** illustrates the process of matching cloud services and building a Semantic Network.

Algorithm 1: Building a Semantic Network

Input: S_{set} is a set of services.
Output: Sem_{net} is a Semantic Network, which includes cloud services with labeled links

```
BuildingASemanticNetwork(Sset)
 1. Semnet ← null;
 2. For  i=0 to i< length(Sset) do
 3.  For  j=1 to j< length(Sset) do
 4.    s1= Sset [i], s2= Sset [j])
 5.     value= Sim(s1, s2)     // finding the matching type of the output of s1 and input of s2
 6.     If  value == 1       // matching type is Exact
 7.       create an edge between s1 and s2 and label the edge with 1, s2 is a successor to s1
 8.       Semnet= Semnet ∪ {<s1,1,s2>}
 9.     elseIf  value == 1/2  // matching type is Plug-in
10.       create an edge between s1 and s2 and label the edge with 1/2, s2 is a successor to s1
11.       Semnet= Semnet ∪ {<s1,1/2,s2>}
12.     elseIf  value == 1/3  // matching type is Subsumed
13.       create an edge between s1 and s2 and label the edge with 1/3, s2 is a successor to s1
14.       Semnet= Semnet ∪ {<s1,1/3,s2>}
15.     else
16.       No edge between s1 and s2 and label the edge is created
17.       Semnet= Semnet ∪ {<s1,0,s2>}
18. return Semnet
```

In **Algorithm 1**, *Sim*(s_1, s_2) provides a value that represents the matching type between services. The result of the algorithm is a set of linked services that form a Semantic Network.

3.2 Forward-Chaining based Algorithm for Finding Composite Services

In order to automatically find a composite service that satisfies the request of a Cloud Service Consumer, a novel algorithm is proposed. The proposed algorithm adopts Forward-Chaining and Depth-First methods[18] to generate composite services. The algorithm takes the input of the request as the start point, and the output of the request as the end point, and then finds all composite services may generate between these points.

A composite service[18] is a set of linked services, in a Semantic Network, that forms a path, which begins with the requested input and ends with the requested output.

Assume there is a semantic network Sem_{net} and a request $R = <ins, outs>$, where *ins* is a set of requested input(s) and *outs* is a set of requested outputs. A set of composite service $CS_{set} = \{path_1, ..., path_u, ..., path_o\}$ where $path_u$ is a composite service which is automatically generated as solutions for the request using Depth-First and Forward-Chaining methods. Each path, which presents a composite service, begins with the requested input and ends with the requested output.

Algorithm 2 shows the main steps of generating composite services by finding paths using Forward-Chaining method.

Algorithm 2: Generating Composite Services

Input: Sem_{set} is a Semantic Network, which includes cloud services with labeled links; *R.input* is the requested input; *R.output* is the requested output.
Output: CS_{set} is a set of composite services that are presented by paths

GeneratingCompositeServices(Semset, R.input, R.output)
1. $CS_{set} \leftarrow$ null;
2. $CS_{set} \leftarrow CS_{set} \cup$ *FindCompositeServicesByLinks*(Sem_{net}, *R.input*, *R.output*, null, CS_{set})
   // no father service at the beginning (null)
3. *return* CS_{set}

Algorithm 2 generates a set of composite services CS_{set}. A function of finding all possible composite service, called *FindCompositeServicesByLinks*, is invoked with several inputs. These inputs are as follows: Sem_{net} is a Semantic Network, which includes cloud services with labeled links; *R.input* is the requested input whereas *R.output* is the requested output. *father* is the up service of the current services, the initial value of *father* is *null*. CS_{set} is an intermediate set of composite services that will be updated by adding new composite services when *FindCompositeServicesByLinks* function is invoked. CS_{set} will be fed back as the final output of invoking *FindCompositeServicesByLinks* function.

Algorithm 3: Find Composite Services By Links

Input: Sem_{set} is a Semantic Network, which includes cloud services with labeled links; *R.input* is the requested input; *R.output* is the requested output; *father* is the up service of the current services; CS_{set} is an intermediated set of composite services
Output: CS_{set} is a final set of composite services

FindCompositeServicesByLinks(Sem_{set}, *R.input*, *R.output* ,*father*, CS_{set})
1. *start* ← *R.input*; // obtain the requested input
2. *end* ← *R.output*; // obtain the requested output
3. $S_{set,} \leftarrow$ *find*($Sem_{net,}$ *start*) // find cloud services in Sem_{net} whose input is similar (Exact, Plug-in,Subsumed) to *start*
4. stack.length=n // build a stack with n length, n is the number of services in S_{set}
5. stack.father←*father* // determine the father of services in stack
6. for each s in S_{set}
7. stack.push← s // store s into stack
8. end for
9. for each *s* in stack do
10. ser=stack.pop→ s // extract s from stack and assign ser to s

```
11.      if ser.output ! = end    // if ser.output is not similar to end
12.        father ←ser
13.        FindCompositeServiceByLinks(Sem_net, ser.output,end, father, CS_set)  // recursion
14.      else
15.        links←null;
16.        links←GetLinksFromEndToStart (ser, R.input, R.output, links) // find links from start
   to end
17.        path← generatePath(links)          // generate a path using links
18.        CS_set ← CS_set U path
19.      end if
20.  end for
```

Algorithm 3 finds the possible composite services in the Semantic Network Sem_{net}. Firstly, *R.input* and *R.output* are assigned to *start* and *end* variables, respectively. Secondly, the services whose input is similar (Exact, Plug-in or Subsumed) to *start* are stored in the set S_{set}. Thirdly, a *stack* is built and its length is assigned to *n* (number of services in S_{set}). The *father* of this *stack* is assigned to *father* value and the services in S_{set} are stored in *stack.* Fourthly, all services in *stack* are extracted one by one. For each extracted service *s, s* is assigned to *ser,* if the output of *ser* is not similar to *end*, then we recursively call the function *FindCompositeServicesByLinks*, otherwise, the function *GetLinksFromEndToStart* is invoked to find the *links* from *R.output* (*end*) to *R.input* (*start*). A *path*, which begins start to end from *R.input* (*start*), is generated by ordering the *links* to by invoking *generatePath*. Finally, *path* is added to CS_{set}.

Algorithm 4: Get Links from End to Start

Input: *ser* is the current service; *R.input* is the requested input; R.output is the requested output; *links* is an intermediated set of links

Output: *links* is the final set of links from End to Start

```
GetLinksFromEndToStart(ser, R.input,R.output,links)
1.  father←Findfather(ser)        // find the father of service ser
2.  if father is not null and ser.output is not similar to R.output // the current service is in the
                                                middle and the output of ser is not similar
                                                (Exact, Plug-in or Subsumed) to R.input
3.     links ← links ∪ <father.output,ser.input>
4.     ser←father
5.     links ← links ∪ GetLinksFromEndToStart (ser, R.input, R.output, links)  // recursion
6.  elseif father is not null and ser.output is similar to R.output // the current service is at the
   bottom
                                                near to R.output and the output of
ser is similar
                                                (Exact, Plug-in or Subsumed) to
R.input
7.     links ← links ∪ <ser.output, R.output>
8.     links ← links ∪ <father.output, ser.intput>
9.     ser←father
10.    links ← links ∪ GetLinksFromEndToStart (ser, R.input, R.output, links) // recursion
11. elseif father is null and ser.input is R.input      // the current service is at the top near to
R.input and the
                                                output of ser is similar (Exact, Plug-in or
Subsumed) to R.input
12.    links ← links ∪ <R.input, ser.input >
13. end if
```

Algorithm 4 finds the links from *R.output* to *R.input*. Firstly, it finds the *father* service of the current service *ser*. Secondly, the position of the current service ser is checked. If *ser* is in the middle (*ser* is not the first service or the last service), the link between *ser*

and *father* and then *ser* is assigned to *father* and the algorithm invokes itself again. If *ser* is in the bottom (*ser* is the last service), the link from *ser* to R.output and the link from *father* to *ser* are obtained and then *ser* is assigned to *father* and the algorithm invokes itself again. If *ser* is in the top (*ser* is the first service), the link between R.input and *ser* is obtained. Finally, the whole links from R.output to R.input are returned as the output of this algorithm. The flow chart of the forward-Chaining algorithm is shown in **Figure 5**.

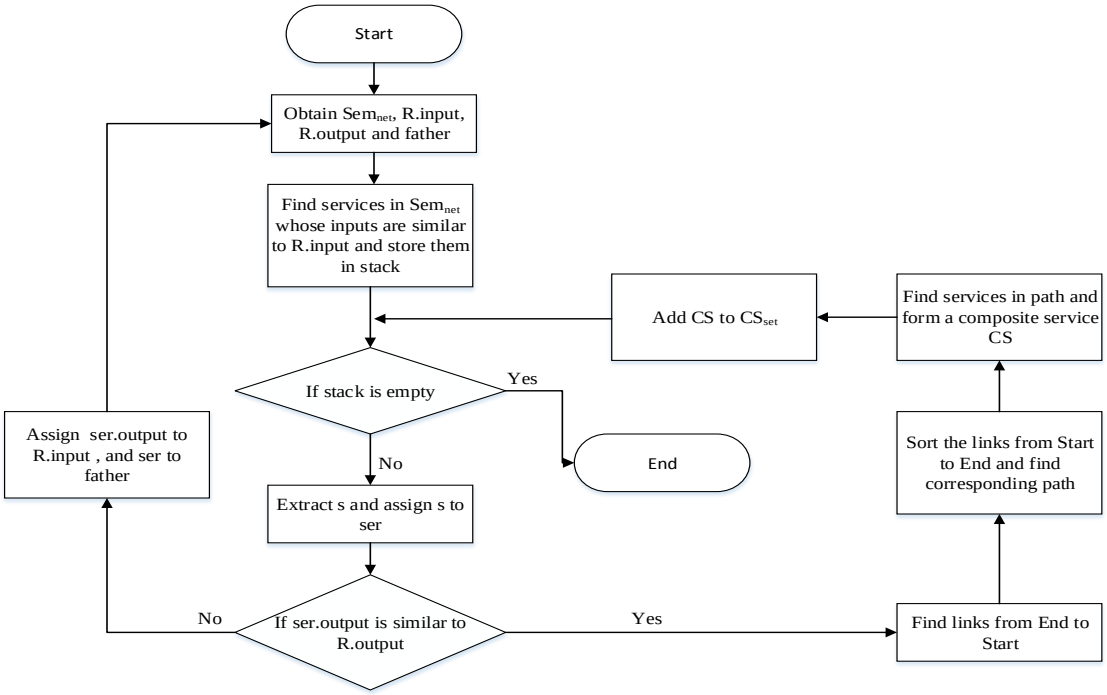

Figure 5. Flow Chart in Forward-Chaining Algorithm

3.3 Hybrid Ranking Method for Selecting Optimal Composite Service

Forward-Chaining based algorithm produces a list of composite services in which each composite service can satisfy the functional requirement of the cloud services consumer. Therefore, a ranking method is required to find the optimal composite service. To this ends, a hybrid ranking method is adopted. This method goes in two steps, Similarity Quality Ranking and QoS-Aware Ranking.

3.3.1 Similarity Quality Ranking

In this step, the Similarity Quality of the each composite service is computed by multiplying the match type values of the links of the path which forms the composite service. The composite service that its path has the highest value is sent back to the service consumer as the optimal composite service. The Similarity Quality of a composite service CS_u, which is formed by $path_u$, is computed as follows.

$$Sim_{quality}(path_u) = Sim(start, s_j.input) \times Sim(s_{j+1}.output, s_{j+2}.input) \times ... \times Sim(s_{j+m-1}.output, end) \quad (1)$$

where *Sim* is the similarity function for obtaining the value matching type of two concepts(the value is 1, 1/2 or 1/3) as in section 3.1, *start* and *end* indicate the concepts of requested input and output of the request respectively. The parameters in *Sim* function are the links that form the $path_u$ of the composite service CS_u.

3.3.2 QoS-aware Ranking

This step is used in the case that the highest value of Similarity Quality is obtained and found by more than one composite service. Thus, QoS-aware ranking [19] is used to find the most suitable composite service, which has the highest Utility Value.

Assume that the Forward-Chaining Algorithm produces a set of composite services $CS_{set} = \{path_1, \ldots, path_u, \ldots, path_o\}$ and each of these paths has the same highest Similarity Quality value. Then, Utility Value is required to rank and find the most suitable path (composite service) in respect of QoS Criteria. As each $path_u$ forms a corresponding composite service CS_u and for the sake for simplicity, in the following we use the concept composite service CS_u instead of $path_u$.

A composite service CS_u, contains n cloud services, such that $CS_u = \{s_j, s_{j+1}, \ldots, s_{j+n-1}\}$, and each service s_j has five QoS Criteria c_k, namely Price, Duration, Availability, Reliability and Reputation as described in the Unified Ontology.

$CS_u.c_k$ indicates the aggregated value [19] of QoS Criteria c_k of a composite service CS_u. $CS_u.c_k$ is computed as shown in **Table 2**.

Table 2. Aggregated Value of QoS Criteria in Sequential Model

QoS Criteria	Price	Duration	Availability	Reliability	Reputation
Aggregated Value	$\sum_{j=1}^{n} s_j.c_k$	$\sum_{j=1}^{n} s_j.c_k$	$\prod_{j=1}^{n} s_j.c_k$	$\prod_{j=1}^{n} s_j.c_k$	$\frac{1}{n}\sum_{j=1}^{n} s_j.c_k$

where s_j is a component service in the composite service CS_u and $s_j.c_k$ indicates QoS criteria value of the s_j.

The utility value [20] , which is denoted by UCS_u, of a composite service CS_u is computed as follows:

$$UCS_u = \sum_{k+} w_{k+} \times \frac{MAX_{k+} - CS_u.c_{k+}}{MAX_{k+} - MIN_{k+}} + \sum_{k-} w_{k-} \times \frac{CS_u.c_{k-} - MIN_{k-}}{MAX_{k-} - MIN_{k-}} \quad (2)$$

where $k+$ indicates a positive QoS criteria such as availability and reputation whereas $k-$ denotes a negative QoS Criteria such as Price and Duration. $CS_u.c_k$ denotes the aggregated *QoS* value of the composite service CS_u, MAX_k and MIN_k indicate the maximum and minimum value of QoS Criteria k of available composite services.

Assume there is a set of composite services $CS_{set} = \{CS_1, \ldots, CS_u, \ldots, CS_o\}$, as each CS_u is formed using a corresponding path $path_u$, then the maximum value of QoS criteria k is computed as $MAX_k = \overset{o}{\underset{u=1}{Max}}(CS_u.c_k)$, and the minimum value of QoS Criteria k is computed as $MIN_k = \overset{o}{\underset{u=1}{Min}}(CS_u.c_k)$.

After computing the Utility Value of all CS_u, which has similar highest value of the Similarity Quality, the composite service that provides the Maximum Utility Value will be fed back to Cloud Service Consumer as the most suitable composite service.

4. Experiment and Analysis

In this section, several experiments are conducted to validate our approach, which achieves better composition efficiency than traditional automatic composition methods, e.g., SAT [21], while finding the optimal composite service using a data set called OWLS-TC [22]. In the following, the content of our experiment is introduced in detail. The content includes Experimental Setting and Experiment Result and Analysis.

4.1 Experiment Setting

4.1.1 Data

The experiments of our work using OWL-TC test collection obtained from http://projects.semwebcentral.org/projects/owls-tc/. With modifying the data set content to be fit with data in the unified ontology, the data set provides around 5000 services and 100 queries (requests) specified with OWL-S in several domains.

All experiments have been performed on a PC using a 4 G memory and i7 CPU. In addition, several programs and software are used such as operating system: Windows 8, Web Server: Tomcat 7.0 and plug-ins: Axis 1.4, Pellet 4.0, Portege 3.1 and Jena 3.2.

4.1.2 Evaluation Method

For the aim of evaluating our experiments, we implemented a scenario, where a request includes a requested input and a requested output. The request of service consumer is solved using the following approaches.

(1) SAT Approach [21]: This approach presents services composition based on the semantic dependency between services using Input-Output matching in order to link these services and then using Casual Link Matrix to build composite services.

(2) Our Approach: This approach includes an Output-Input algorithm for building Semantic Network, Forward-Chaining algorithm, and Hybrid ranking method.

4.2 Experiment Results and Analysis

In this section, we compare and analyze the performance of composition using our approach and the approach in SAT in terms of execution time and Successful Rate.

4.2.1 Successful Rate

Successful Rate of composition is the rate of the ability of a specific approach to successfully finding the solution (composite service) of a given request. In order to evaluate the Successful Rate of our approach and SAT approach, the testing dataset was grouped into five groups as follows: 200, 300, 400, 500, 1000 and 100 requests were provided to find solution (composite service). **Figure 6** shows the Successful Rate of the two approaches in respect with the five groups of testing dataset.

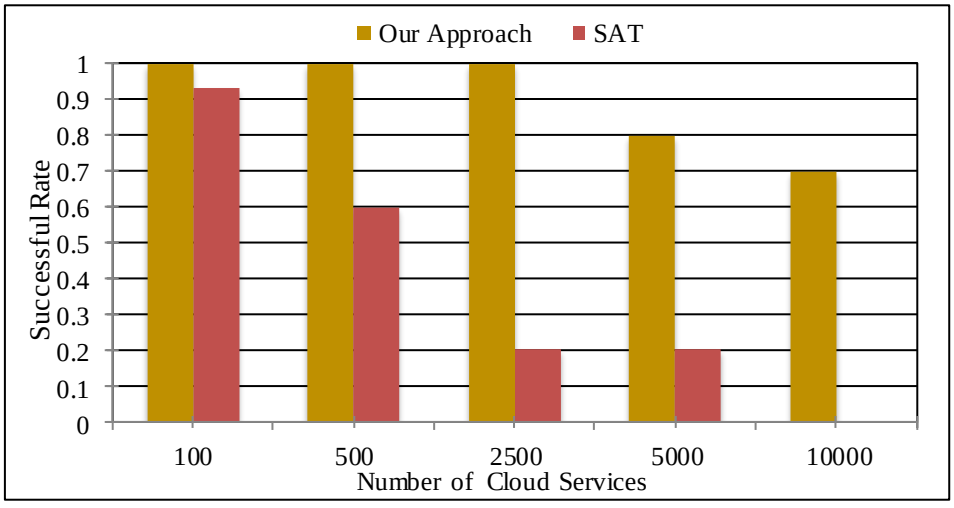

Figure 6. Comparison of the Successful Rate of the Two Approaches

Since a testing dataset included small number of services the two approaches find solutions, but since a testing dataset included big number of services SAT performs low Successful Rate and reached 0 since reached 10000 services. While our approach had a

low effect since the number of services in a testing dataset went small, and this means that performs better than SAT and has a high ability to find solution for requests.

4.2.2 Execution Time

In this experiment, the execution time was studied using varied numbers of cloud services. The numbers of services were varied from 200 to 1000. The execution time indicates the consuming time for achieving a request of the service consumer. **Figure 7** shows the execution time in terms of varied numbers of cloud services.

As **Figure 7** shows, SAT was faster as small number of services whereas our approach was faster as large number of cloud services. The execution time of both approaches increased as the number of cloud services increased. However, the execution time of our approach degraded more gracefully than in SAT as building a semantic network, in our approach, is an offline process thus the consuming time of building a semantic network is trivial and not considered. **Figure 8** shows the execution time for SAT and our approach in the case that the number of services is fixed to 1000 and the number of requests is varied from 20 to 100. As **Figure 8** shows, the gap in execution time between SAT and our approach is widened as the number of requests increased. This proves that our approach scales better than the SAT approach as the number of requests increases.

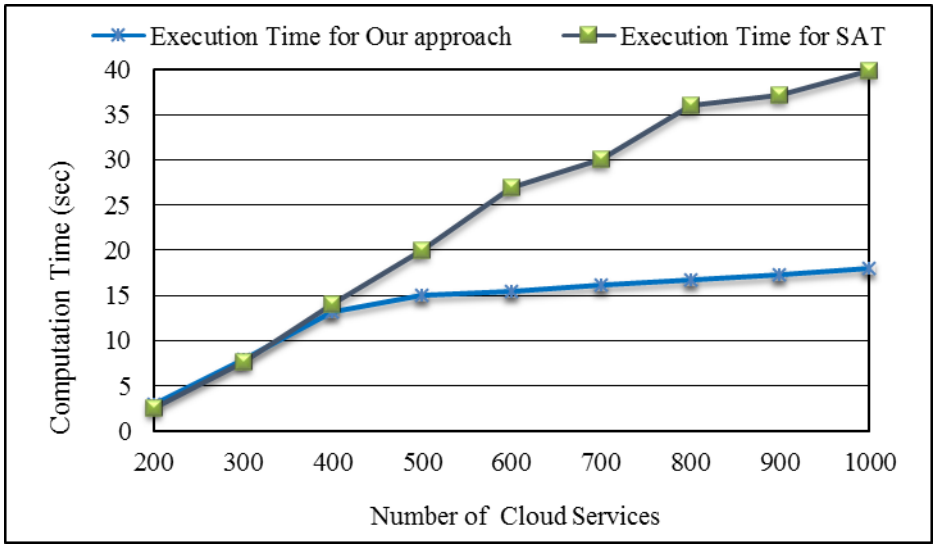

Figure 7. Execution Time Comparison in Term of Varied Number of Services

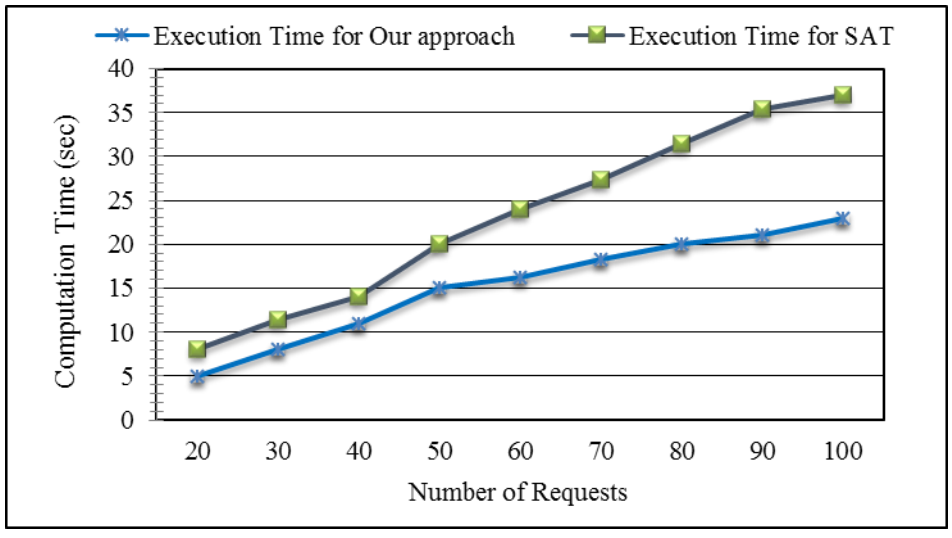

Figure 8. Execution Time Comparison between Our Approach and SAT

4.2.3 QoS Optimality

Our approach adopted QoS-based Ranking method to select the optimal composite service according to the requirements of services consumer. This enhanced the successful rates and efficiency of composition as selecting the best composite service among a set of composite services that satisfies the functional requirement. SAT [21] ignores the important role can QoS Criteria play for ranking the available composite services.

5. Conclusions

The ability of Cloud Services to compose and integrate loosely coupled applications has attracted the attentions of researchers in the field of Automatic Cloud Services Composition. This paper proposed a semantic-aware approach for automatic Cloud Services Composition. The proposed approach contains three steps. Firstly, an inter-connected network (called semantic network) is built using an efficient approach of the similarity (Outputs-Inputs similarity) among semantic concepts of cloud services. Secondly, a Forward-Chaining method is adopted to create composite services that functionally satisfy the Service Consumer's request (requested inputs and requested outputs). Thirdly, as several composite services may be generated from the Forward-Chaining method, a hybrid ranking method based on similarity quality and QoS Criteria is adopted and used to select " the suitable " composite service for the Service Consumer.

In this papers, more attention was paid on both aspects: 1) building a semantic network, which is an offline process and then its consuming time is trivial, 2) performing ranking composite services process which selects the suitable composite service using services consumer's preferences.

It is clear from the above results that our approach significantly outperforms the traditional approach SAT on the consuming time of services composition and provides an efficient ranking method for generated composite. The study results clearly suggest that using our approach may be a better choice for automatic services composition.

Future work may involve services composition for other types of cloud services (such as IaaS) and may consider designing and implementing methods and models taking into account computing resources such as CPU cycles, data center space, storage resources.

Acknowledgement

This study is sponsored by the National Key Technology R&D Program (2014BAG01B03), the National Natural Science Foundation of China (51105286, 61203236) and the Academic Leader Project of Wuhan (2012711304457) and the Fundamental Research Funds for the Central Universities (WUT: 2014-IV-137).

References

[1] Leitner, Philipp, Waldemar Hummer, Schahram Dustdar. "Cost-based optimization of service compositions", IEEE Transactions on Services Computing. Vol. 3, no. 2,(**2013**), pp. 239-251.
[2] Joshi K.P, Yesha Y, Finin T. "Automating Cloud Services Life Cycle through Semantic Technologies". IEEE Transactions on Services Computing", vol.7, no.1,(**2014**) pp. 109-122.
[3] Quan Z Shenga, Xiaoqiang Qiao, Athanasios V Vasilakosc, Claudia Szaboa, Scott Bournea, Xiaofei Xu. "Web services composition: A decade's overview". Information Sciences. vol. 28, (**2014**), pp. 218–238.
[4] Eduardo Gonçalves da Silva, Luís Ferreira Pires,Marten van Sinderen." Towards runtime discovery, selection and composition of semantic services". Computer communications. vol.34, no.2, (**2011**).pp. 159-168.
[5] Martín Serrano, Lei Shi, Mícheál Ó Foghlú, William Donnelly. "Cloud services composition support by using semantic annotation and linked data. Knowledge Discovery". Knowledge Engineering and Knowledge Management. (**2013**),pp. 278-293.
[6] Wei Tan, Yushun Fan, MengChu Zhou, Zhong Tian. "Data-driven service composition in enterprise SOA solutions: a Petri net approach". IEEE Transactions on Automation Science and Engineering. vol.7, no.3, (**2010**), pp. 686-694.

[7] Bartalos, P, Bieliková M. "QoS aware semantic web service composition approach considering pre/postconditions". In Web Services (ICWS), 2010 IEEE International Conference on, (**2010**),pp.345-352.
[8] Omer A M, Schill A. Dependency based automatic service composition using directed graph. Next Generation Web Services Practices, NWESP'09. Fifth International Conference on. IEEE, 2012, pp. 76-81.
[9] Noor T.H, Sheng Q.Z , Ngu, A.H.H, Dustdar S."Analysis of Web-Scale Cloud Services". IEEE Internet Computing. vol.18, no.4,(**2014**), pp.55-61.
[10] Lars Nielsen. "The Little Book of Cloud Computing: Including Coverage of Big Data Tools". New Street Communications, LLC.(**2013**).
[11] Afify, Yasmine M.."Cloud Services Discovery and Selection: Survey and New Semantic-Based System". Bio-inspiring Cyber Security and Cloud Services: Trends and Innovations. Springer Berlin Heidelberg, (**2014**). pp.449-477.
[12] Bernardo Cuenca Grau., et al. "OWL 2 Web Ontology Language: Profiles (Second Edition)" .http://www.w3.org/TR/owl2-profiles/.(**2015.12.11**).
[13] Afify, Yasmine M., et al. "Concept Recommendation System for Cloud Services Advertisement. Advanced Machine Learning Technologies and Applications". Springer International Publishing,(**2014**). pp.57-66.
[14] Zou G, Lu Q, Chen Y, et al. "QoS-aware dynamic composition of Web services using numerical temporal planning". Services Computing, IEEE Transactions on. vol.7, no.1, (**2014**),pp. 18-31.
[15] Hasan Naji,A.H, Gao Shu, Al-Gabri Malek. "An optimal semantic network-based approach for web service composition with qos", Telkomnika. vol.11, no 8, , (**2013**),pp. 4505-4511.
[16] Cassar G, Barnaghi P, Wang W, et al. "A hybrid semantic matchmaker for IoT services". Green Computing and Communications (GreenCom), 2012 IEEE International Conference on. IEEE, (**2012**), pp. 210-216.
[17] Dong H, Hussain F K, Chang E. "Semantic Web Service matchmakers: state of the art and challenges. Concurrency and Computation: Practice and Experience". vol.25,no.7, (**2013**), pp. 961-988.
[18] Khakhkhar S, Kumar V, Chaudhary S. "Dynamic Service Composition". International Journal of Computer Science and Artificial Intelligence. vol. 2, no.3, (**2012**), pp. 32-42.
[19] Zheng, Zibin, et al. "QoS ranking prediction for cloud services". IEEE Transactions on Parallel and Distributed Systems, vol.24, no.6, (**2012**), pp. 1213-1222.
[20] M. Alrifai, T. Risse, W. Nejdl. "A hybrid approach for efficient web service composition with end-to-end QoS constraints". ACM Transactions on the Web. vol.6, no.2, p. 7. http://dx.doi.org/10.1145/2180861.2180864.
[21] Kil, Hyunyoung, Wonhong Nam. "SAT Solving Technique for Semantic Web Service Composition. Computer Applications for Web, Human Computer Interaction, Signal and Image Processing, and Pattern Recognition". Springer Berlin Heidelberg. (**2012**), pp. 167-172.
[22] http://projects.semwebcentral.org/projects/owls-tc/, (**2016.01.15**).

Authors

Hasan.A.H Naji, He is working as an Assistant Professor at Wuhan University of Technology, Wuhan, China. He has received her Master Degree and PhD in Computer science and Technology from Wuhan University of Technology, Wuhan, China. His areas of research are Service-Oriented Computing, Transportation data analysis, and Data mining.

Chao Zhong Wu, He is working as a Fulltime Professor at Wuhan University of Technology, Wuhan, China. He has received her Master Degree and PhD in Transportation Engineering from Wuhan University of Technology, Wuhan, China. His areas of research are Transportation data analysis, Driver behavior and Driver safety.

Shu Gao, She is working as a Fulltime Professor at Wuhan University of Technology, Wuhan, China. She has received her Master Degree and PhD in Computer Science from Wuhan University of Technology, Wuhan, China. Her areas of research are Data mining and Transportation data analysis.