A Methodology to Compose Web Services using Compatible Components based on QoS and Security Requirements of the Users

Urjita Thakar
Associate Prof,
Deptt. of Comp. Engg.,
Shri G.S.Inst. of Tech. & Sc.,
23, Sir Visweswaraiya Road,
Indore (MP) INDIA 452003

Nirmal Dagdee
Director & Professor CSE,
S.D.B. College of Tech.
A.B. Road,
Umaria, Indore (MP)

Abhishek Agrawal
Department of Comp.Engg.
Shri G.S. Inst. of Tech. & Sc.,
23, Sir Visweswaraiya Road,
Indore (MP) INDIA 452003

ABSTRACT
Many online businesses offer services to the users that are complex in nature. A number of simple services need to be combined to form a composite service. User demands may include not only complex functional requirements but may also pertain to specific QoS and security needs. The overall QoS of the composite service and the offered security may be decided by the QoS and the security support given by each individual constituting component. In this paper, a methodology has been proposed to form complex service sets using component services with matching QoS values that shall also offer a minimal of security support satisfying user's requirement. Availability, response time and throughput are important QoS parameters. Taking into account, the user's requirements related to confidentiality, integrity and authentication, service sets are presented to the user in the form of a list such that the set with highest level of compatibility appears at the top. The proposed method thus is very useful for the user and enables him to avail the complex service that suit the best to his needs.

General Terms: Composite Web services

Keywords: Composite Web Services, Compatibility, Security, QoS

1. INTRODUCTION
The number of web services available on the Internet has increased exponentially. Many such services are complex in nature. Multiple simple services need to be combined to form such complex services. To meet the expectations of the customer, the component services constituting them should be in harmony with each other. Different services are offered with different values for quality of service (QoS) parameters and different security support by various providers. Some providers may offer high value for a particular QoS parameter, while some other providers may offer lower value for the same parameter. The overall QoS of the composite service may be decided by the worst QoS value of a constituting component. Availability, response time and throughput are QoS parameters indicating performance of a service and are important for providers as well as requesters. Service requesters are highly concern about the security related issues as services are accessed in open environment. They wish to access the services that shall fulfill the required security needs. Corresponding to different security functions, a number of algorithms with varying strengths exist. Different service providers may apply different algorithms to provide the same security function. The strength of the algorithm decides the strength of the offered security to the user. The strength of the security of the offered complex service corresponds to the weakest algorithm used by a constituting component service.

Presence of large number of services, with diverse values of QoS and the offered security support, is a challenge for a business to offer a complex service that shall fulfill not only the desired functional need but shall also provide adequate quality of service while guarantying the minimal requisite security. Some earlier work present in the literature pertaining to composition of services based on user's constraints was presented by Senkul et al.[1]. However, it does not take care of the QoS and security related needs of the user.

In this paper, a method has been presented to compose web services using component services with matching QoS values that shall offer the minimal security support satisfying user's requirements.

Rest of the paper is organized as follows: Related work is discussed in section 2. In section 3, the proposed method is discussed and the proposed algorithms are discussed in section 4. In section 5, testing and results of the presented approach have been discussed. Concluding remarks and the future work are given in section 6.

2. RELATED WORK
Some earlier research work carried out by various researchers relevant to the field is surveyed and discussed in this section.

In the work proposed by Ming et al., a solution for dynamic web service composition is discussed in which user's requirement is broken down into a series of abstract web services [2]. Carminati et al. have proposed a solution to achieve dynamic service composition. In the work, the BPEL document describing the composite service is modified to add some compatibility checking constraints [3]. The method for dynamic web service composition proposed by Wang et al. is useful in finding matching serices using semantic information.[4]. An architecture for dynamic composition of web services as per user's requirements and availability of resources has been proposed by Boumhamdi et al.[5].

Al-Masri et al. developed Web Service Relevancy Function (WsRF) for measuring the relevancy ranking of a particular

Web service based on client's preferences and QoS metrics [6]. The ranking function finds the best available Web service during service discovery process based on a set of given client QoS preferences. Thakar et al. have proposed an approach to enable a service provider to publish the information related to the security support offered and to facilitate the service requester to discover the providers as per the security related needs [7].

A composition method taking care of user preferences in which the services are described using OWL-S has been presented by Naiwen Lin et al. that uses AI based technique [8]. Another method for service selection and composition has been presented by Matskin et al. which uses agents [9]. The method is based on logic and uses semantic reasoning. By semantic matching among these abstract services, a service composition is obtained for execution. In the work presented by Liu et al., a compatibility checking algorithm based on the concept of equivalence has been presented [10]. The paper discusses compatibility at different levels and its impact on composition and substitution of Web services. An architecture based on agents for evaluating web service QoS parameters has been proposed by Thirumaran et al. [11]. The paper discusses various useful QoS parameters for web services.

3. PROPOSED METHOD

In the proposed method, the online business offers the complex service to the user through an agent. The agent takes user's requirements for the desired functionality, security functions and the cost that he is ready to pay for the QoS and the security while the complex service is availed. Architecture of the proposed system is shown in Fig. 1.

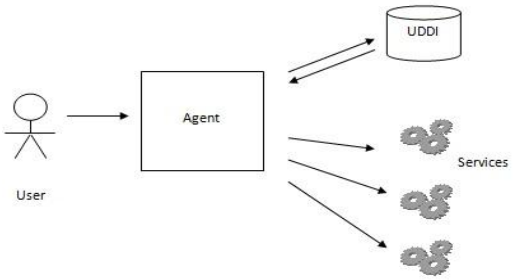

Fig. 1: System Architecture

Agent is the key module in the proposed architecture. Based on user's requirement for service functionality obtained through the user interface, it discovers the services from the registry along with security related information. Values of QoS parameters for all the component services are calculated by the agent. For the desired complex service, it then finds the component service combinations that contain services that are compatible to each other with regard to the QoS values also meeting the cost requirement of the user. These service sets are then ranked based on the security support of the service set such that set with high security strength appears at the top. Architecture of the agent is shown in Fig. 2.

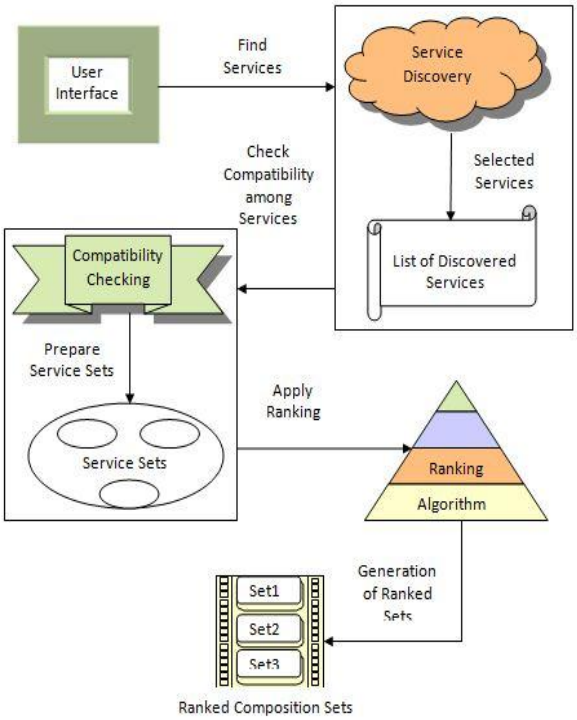

Fig. 2: Architecture of Agent

The modules present in the agent are discussed below.

1. User Interface- This module takes user's need related to functionality, security and the cost he is ready to pay to avail the service.

2. Service Discovery Module- This module discovers component services based on user's functional requirements along with the offered security support and cost from registry. It prepares a list of discovered component services required for composition.

3. Compatibility Checking Module- In this module the compatibility among component services based on cost and QoS parameters is checked. It prepares the sets of compatible component services.

4. Ranking Module- This module ranks the service sets based on the security offered by the service set and the cost such that set offering highest security strength appears at the top.

In the next subsection, the method for calculation of values for QoS parameters is discussed.

3.1 Calculation of Values for QoS Parameters

In this work, three important QoS parameters availability, response time and throughput are considered. The agent calculates the values for these QoS parameters for each service provider as discussed below and stores in a database.

Availability: To find the availability of services, agent takes endpoint URL of a service from the service registry and generates a request for each service. If a reply is received in expected time then that service is considered to be available. The number of successful and total invocations are counted for each service for a given time interval. Availability of the service is calculated using following equation.

Availability = Number of successful invocations / Total invocations

Response Time: To find the response time of a service, agent notes the time of sending the request and arrival of response. The response time of the service is calculated as given below.

Response Time = Time of receiving Response – Time of sending the request

Throughput: To find the throughput of services, agent generates a number of requests for a particular service for a fixed time period. The corresponding responses from the service are captured. Throughput of the service is calculated using following equation.

Throughput = Total number of handled requests / time

Performance of a service with high availability, low response time and high throughput is considered better than others.

The steps followed to generate list of compatible component service sets are given below.

Step 1: Agent accepts functional, security and cost related requirements of the services from the user.
Step 2: Agent searches the UDDI registry to find the services based on the requirement.
Step 3: Values of QoS parameters for each service are calculated as discussed in previous section and are stored in the database. The compatibility of the discovered services is checked using their QoS parameter values by applying a 'compatibility checking algorithm' described in section 4.1. Service sets containing compatible component services are also generated.
Step 4: A ranking algorithm described in section 4.2 is applied on the service sets to arrange these sets in an order such that set with highest security strength with the desired requirement appears at the top.
Step 5: The component services present in the service set are executed as per the business process.
Step 6: If at the time of service execution, any component service fails then a rollback is performed and the next highest ranked set is selected and the execution of its services is started.
Step 7: This process is repeated till either a complete set is executed or all the sets have been tried.
Step 8: Final result is displayed to the user.

The algorithms proposed to determine compatibility of services and ranking of the service sets are presented in the next section.

4. PROPOSED ALGORITHMS

In this section, the algorithm proposed for obtaining sets of compatible component services and ranking these sets are discussed. The algorithm for checking compatibility among component services based on QoS parameters is discussed next.

4.1 Compatibility Checking Algorithm

The compatibility checking algorithm works in two steps. In the first step, complex service sets satisfying cost constraints are obtained. In the second step, compatibility among component services of each complex service set is checked based on QoS parameters.

In the first part of algorithm, maximum cost value payable for a complex service is used as threshold value to shortlist the complex services. In first iteration, two services with different functionalities that appeared in the business process are considered for service set and total cost of this service set is compared with the threshold. If it is less than or equal to the threshold cost then that service set is considered to be partially accepted, otherwise rejected. In the next iterations accepted partial service sets are combined with other services with different functionality one by one to obtain more such service sets. Total cost of these service sets is compared with the threshold cost. These iterations are repeated until complex service sets containing all required component services of different functionalities for desired business process are generated.

In next part of the algorithm, to determine compatibility among component services of complex service sets, maximum acceptable difference of QoS values defined by the administrator is used as threshold. All component services with required functionality that have acceptable difference in QoS values are included in the set of compatible service, otherwise rejected.

QoS value of a service set can be defined by the weakest QoS value of that set, therefore this algorithm calculates the weakest QoS value for all compatible service sets and arranges all sets in descending order of the offered QoS.

Input: Number of services, Number of Service Providers for each service, Service Providers for each type of service, User's Readiness to pay, Cost charged by each service provider, QoS parameter values for each Service Provider, Maximum acceptable difference in QoS values, Required number of composite service sets.
Output: Compatible Component Service Sets

Algorithm:

```
CompatibilityCheck()
  Declare arrays arr1[], arr2[] and arr3[]
  //arr1[], arr2 and arr3[] stores service sets.
  set: arr1[] = All Service Providers of 1st type of service.
  for (i = 2 to Number of Services to be composed)
      set: arr2[] = All Service Providers of ith type of
      service   for (j = 1 to number of service sets in arr1[])
                    for( k = 1 to number of service providers
                      in arr2[])
                      arr3[] = arr1[j] + arr2[k]  //Generates
                      new service set by combining service
                      providers of arr1[j] with service
                      providers of arr2[k]. //
    End of for loop // k
  End of for loop //Gets number of service sets. Each
      service set is combination of i type of services.
      clear arr1[]//Delete data stored in arr1[].
          arr1[] = MaxCost(arr3[])//Checks service sets
          stored in arr3[] that are satisfying user's readiness
          to
  pay and stores in arr1[].
          clear arr3[] and arr2[] //Delete data stored in
      arr3[] and arr2[].
      End of for loop //i  //Generates service sets having
      Service Providers of all types and satisfying user's
      readiness to pay.//
  arr3[]: MinQoS(arr1[])//Checks compatibility based on
      QoS parameters among component services of each
      service set.
```

for (i = 1 to Required number of composite service sets)
final[i] = arr3[i]
End of for loop//Required number of Compatible Composite service sets with highest QoS values are resulted.

MaxCost(arr3[] ServiceSets)
Declare variable arr1[].//arr1[] contains service sets satisfying user's readiness to pay.
for (i = 1 to Number of service sets in arr3[])
if (total cost of service set arr3[i] <= Readiness to pay) then
arr1[] = arr3[i]//Service set arr3[i] is satisfying cost constraints, therefore stored in arr1[].
End of if
End of for loop//Checks cost constraints for all service sets.
return arr1[]

MinQoS(arr3[]
ServiceSetsSatisfyingUser'sReadinessToPay)
Declare variable flag, arr1[], arr2[][], qos[]
//arr1[] stores compatible service sets, arr2[][] stores component service providers of each service set, qos[] stores QoS value of service sets.
for (i = 1 to Number of service sets in arr3[])
set: flag = 0
Parse component service providers from service set arr3[i]
for (j = 1 to Number of Component service providers in service set arr3[i])
arr2[i][j] = j^{th} component service provider of service set arr3[i]
End of for loop
for (j = 1 to Number of Component service providers in service set arr2[i][])
for (k = j+1 to Number of Component service providers in service set arr2[i][])
if (abs(QoS parameter value of service provider arr2[i][j] – QoS parameter value of service provider arr2[i][k]) > Maximum acceptable difference in QoS values) then set: flag = 1//Difference in QoS parameter values of service providers of service set arr3[i] is greater than maximum acceptable difference in QoS values.
End of if
End of for loop
End of for loop
if (flag = = 0) then
arr1[] = arr3[i] //Difference in QoS parameter values of all service providers of service set arr3[i] is less than maximum acceptable difference in QoS values. Therefore, accepted as Compatible service set.
qos[] = min(QoS parameter values of all component services of service set arr2[i][])
End of if
End of for loop
for (i = 1 to Number of service sets in arr1[])
for (j = i+1 to Number of service sets in arr1[])
if (qos[i] < qos[j]) then
//Selection Sort is applied to arrange service sets in descending order of their QoS values.
Exchange arr1[i] with arr1[j]
Exchange qos[i] with qos[j]
End of if
End of for loop
End of for loop //Get Compatible service sets in descending order of their QoS values.
return arr1[]

4.2 Ranking Algorithm

For each of the acceptable service sets, the security strength is determined. Security strength of a service set for a specific security constraint is defined by weakest algorithm of constituting component services. All service sets are arranged as per their security strength such that set offering highest security strength appears first. If security strength of two sets is same, then set charging less is ranked higher. The service sets offering highest security are presented to the user. The algorithm is discussed below.

Input: Compatible Component Service Sets (final[])
Output: Ranked Compatible Component Service Sets (final[])

Algorithm: Rank()
Declare variable arr[][], strength[i]
//arr[][] stores component service providers of each service set, strength[] stores security algorithm for each service set.
for (i = 1 to Number of service sets in final[])
Parse component service providers from service set final[i]
for (j = 1 to Number of Component service providers in service set final[i])
arr[i][j] = j^{th} component service provider of service set final[i]
End of for loop
for (j = 1 to Number of Component service providers in service set arr[i][])
if (j = = 1) then
strength[i] = Algorithm supported by 1^{st} component service provider of service set final[i][
else if (strength[i] contains better security algorithm for service set arr[i][] than algorithm supported by j^{th} component service provider of service set arr[i][]), then
strength[i] = Algorithm supported by j^{th} component service provider of service set arr[i][]
endif
End of for loop
End of for loop //strength[] stores weakest algorithm supported by component service providers of service set final[]
for (i = 1 to Number of service sets in final[])
for (j = i+ 1 to Number of service sets in final[])
if (strength[i] < = strength[j]) then
//Security algorithm for service set final[i] is weaker or equivalent to the Security algorithm for service set final[j].
if (strength[i] = = strength[j]) then
//Security algorithm for service set final[i] is equivalent to the Security algorithm for service set final[j]. Therefore, security cost of service set final[i] and final[j] are compared.
if (security cost of service set final[i] > security cost of service set final[j]) then
//Security cost of service set final[i] is greater than security cost of service set final[j]. Therefore, values of final[] and strength[] at i^{th} position is exchanged with j^{th} position.

Exchange final[i] with final[j]
Exchange strength[i] with strength[j]
End of if
else //Else for step 18 if Security algorithm for service set final[i] is weaker than the Security algorithm for service set final[j]. Therefore, values of final[] and strength[] at i^{th} position is exchanged with j^{th} position.
Exchange final[i] with final[j]
Exchange strength[i] with strength[j]
End of for if
End of if
End of for loop
End of for loop

The working of the proposed system is tested and results are discussed in the next section.

5. TESTING AND RESULTS

To test the viability and usefulness of the proposed method, a complex tour planning service consisting of Travel, Hotel and Pickup is considered. In the experiment, 15 Travel Services, 10 Hotel services and 10 Pickup services were deployed. User's requirements were collected through a user interface to obtain the desired component services, required security parameters for those services and his willingness to pay.

Let the requirements given by the user for the three component services be as follows-

For Travel Service:
Confidentiality: Yes
Integrity
Authentication: Yes

For Hotel Service:
Confidentiality: Yes
Integrity
Authentication: Yes

For Pickup Service:
Confidentiality : Yes
Integrity : No
Authentication: Yes

Ready to pay the cost: Atmost 68 units.

The service providers that offer different security algorithms are discovered from the registry. In Table1, the service providers discovered from the registry and the algorithms supported by them are given.

Table 1: Service Providers Discovered From the Registry

S. No.	Provider	Supported Algorithms	Total Cost
1.	TravelA	3DES + DigitalSignatureWithTimeStamp	13
2.	TravelA	3DES + DigitalSignature	9
3.	TravelA	RC6 + DigitalSignatureWithTimeStamp	15
4.	TravelA	RC6 + DigitalSignature	11
5.	TravelA	MARS + DigitalSignatureWithTimeStamp	17
6.	TravelA	MARS + DigitalSignature	13
7.	TravelA	AES128 + DigitalSignatureWithTimeStamp	19
8.	TravelA	AES128 + DigitalSignature	15
9.	TravelA	AES192 + DigitalSignatureWithTimeStamp	21
10.	TravelA	AES192 + DigitalSignature	17
11.	TravelA	AES256 + DigitalSignatureWithTimeStamp	23
12.	TravelA	AES256 + DigitalSignature	19
13.	TravelB	3DES + DigitalSignatureWithTimeStamp	13
14.	TravelB	3DES + DigitalSignature	8
15.	HotelA	AES192 + DigitalSignatureWithTimeStamp	22
16.	HotelA	AES192 + DigitalSignature	18
17.	HotelA	AES128 + DigitalSignatureWithTimeStamp	17
18.	HotelA	AES128 + DigitalSignature	13
19.	HotelA	3DES + DigitalSignatureWithTimeStamp	14
20.	HotelA	3DES + DigitalSignature	10
21. : No	HotelB	3DES + DigitalSignatureWithTimeStamp	13
22.	HotelB	3DES + DigitalSignature	9
23. : No	HotelB	RC6 + DigitalSignatureWithTimeStamp	15
24.	HotelB	RC6 + DigitalSignature	11
25.	HotelB	MARS + DigitalSignatureWithTimeStamp	17
26.	HotelB	MARS + DigitalSignature	13
27.	HotelB	AES128 + DigitalSignatureWithTimeStamp	19
28.	HotelB	AES128 + DigitalSignature	15
29.	PickupA	3DES + DigitalSignatureWithTimeStamp	13
30.	PickupA	3DES + DigitalSignature	8
31.	PickupA	RC6 + DigitalSignatureWithTimeStamp	15
32.	PickupA	RC6 + DigitalSignature	10
33.	PickupA	MARS + DigitalSignatureWithTimeStamp	17
34.	PickupA	MARS + DigitalSignature	12
35.	PickupA	AES128 + DigitalSignatureWithTimeStamp	19
36.	PickupA	AES128 + DigitalSignature	14
37.	PickupA	AES192 + DigitalSignatureWithTimeStam	21

		p	
38.	PickupA	AES192 + DigitalSignature	16
39.	PickupA	AES256 + DigitalSignatureWithTimeStamp	23
40.	PickupA	AES256 + DigitalSignature	18
41.	PickupB	AES256 + DigitalSignatureWithTimeStamp	25
42.	PickupB	AES256 + DigitalSignature	21

As per user's need, 2 services for Travel service, 2 services for Hotel service and 2 services for Pickup service with different combinations of security algorithms were discovered from the registry.

If the travel service is obtained from the service provider 'TravelB' that offers security using AES256 and DigitalSignatureWithTimeStamp algorithms, hotel service is obtained from HotelG that offers security using algorithms AES256 and DigitalSignatureWithTimeStamp and the pickup service is obtained from the service provider PickupB that offers security using algorithms AES256 and DigitalSignatureWithTimeStamp, then the total cost for security of this service set is 73 units which is greater than the user's readiness to pay. Therefore, this service set is discarded. Similarly, when TravelF with security support AES192 and DigitalSignatureWithTimeStamp, HotelD with security support AES192 and DigitalSignatureWithTimeStamp and PickupK with security support AES192 and DigitalSignatureWithTimeStamp are composed then the total cost for security of this service set is 65 units which is less than the user's readiness to pay. Therefore this service set is accepted.

Compatibility among component services of composite service sets is determined based on QoS parameter values. Four important QoS parameters namely availability, response time, and throughput are considered in this paper. Table 2 shows the QoS values calculated for the discovered service providers. In Table 3, the normalized QoS values are shown.

Table 2: QoS Values for Discovered Service Providers

Service Provider	Response Time	Availability	Through put
TravelA	18101	91.6667	797
TravelB	20760	91.6667	849
TravelC	23000	91.5254	882
TravelD	16081	91.5254	833
TravelF	14678	91.5254	895
TravelK	22000	91.0714	900
TravelL	16037	90.9091	863
TravelM	17000	90.9091	754
TravelN	18167	90.9091	770
TravelO	19000	90.9091	768
HotelA	19036	81.4634	234
HotelD	18000	85.2055	595
HotelF	20000	92.8571	479
HotelG	20000	92.6471	462
HotelH	18036	92.5373	478
HotelJ	20000	92.5373	502
PickupA	20029	92.5373	476
PickupB	17066	92.5373	509
PickupC	25000	92.5373	473
PickupH	20031	91.9355	731
PickupI	20038	91.9355	842
PickupK	18060	91.9355	833

Table 3: Normalized QoS Values

Service Provider	Response Time	Availability	Through put
TravelA	0.8108	1	0.8855
TravelB	0.7070	1	0.9433
TravelC	0.6381	0.9984	0.98
TravelD	0.9127	0.9984	0.9255
TravelF	1	0.9984	0.9944
TravelK	0.6671	0.9935	1
TravelL	0.9152	0.9917	0.9588
TravelM	0.8634	0.9917	0.8377
TravelN	0.8079	0.9917	0.8555
TravelO	0.7725	0.9917	0.8533
HotelA	0.9455	0.8772	0.3932
HotelD	1	0.9175	1
HotelF	0.9	1	0.8050
HotelG	0.9	0.9977	0.7764
HotelH	0.9980	0.9965	0.8033
HotelJ	0.8181	0.9965	0.8436
PickupA	0.8520	1	0.5653
PickupB	1	1	0.6045
PickupC	0.6826	1	0.5617
PickupH	0.8519	0.9934	0.8681
PickupI	0.8516	0.9934	1
PickupK	0.9449	0.9934	0.9893

In this test case, total 180000 composite service sets were generated. Based on the average normalized QoS values of all the component services for each set, 30 composite service sets with highest QoS values are considered.

The agent ranks these service sets based on the offered QoS as shown in Table 4. To rank the service sets, the agent checks security strength of each set. Security strength of a service set depends on security algorithm supported by the component services for a particular security constraint. The weakest algorithm for that particular security constraint shall decide the security strength for required security constraint of that service set. The service set with strong algorithm and low cost is ranked better. In this test case, service set containing services TravelL with security support AES256 and DigitalSignatureWithTimeStamp, HotelD with security support AES256 and DigitalSignatureWithTimeStamp and PickupK with security support AES192 and DigitalSignatureWithTimeStamp offer best security costing 67 units, thus it appears at the top.

Table 4: Ranked Compatible Component Service Sets

Provider	Supported Algorithms	Total Cost	Strength
TravelL+HotelD+PickupK	AES256+DigitalSignatureWithTimeStamp+AES256+DigitalSignatureWithTimeStamp+AES192+DigitalSignatureWithTimeStamp	67	AES192+DigitalSignatureWithTimeStamp
Trave	AES192+DigitalSignatureWithTimeSt	6	AES192+

lF+H otelD +Pick upH	amp+AES256+DigitalSignatureWithTi meStamp+AES256+DigitalSignature WithTimeStamp	8	DigitalSig natureWit hTimeSta mp
Trave lL+H otelD +Pick upK	AES256+DigitalSignatureWithTimeSt amp+AES192+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	65	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upH	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	66	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upH	AES192+DigitalSignatureWithTimeSt amp+AES192+DigitalSignatureWithTi meStamp+AES256+DigitalSignature WithTimeStamp	66	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelH +Pick upI	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	67	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upI	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	67	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upK	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	67	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelH +Pick upK	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	67	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelH +Pick upI	AES192+DigitalSignatureWithTimeSt amp+AES192+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	65	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelF +Pick upK	AES192+DigitalSignatureWithTimeSt amp+AES192+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	65	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upK	AES192+DigitalSignatureWithTimeSt amp+AE192+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	65	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelH +Pick upK	AES192+DigitalSignatureWithTimeSt amp+AES192+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	65	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upI	AES192+DigitalSignatureWithTimeSt amp+AES192+DigitalSignatureWithTi meStamp+AES192+DigitalSignature WithTimeStamp	65	AES192+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upH	AES192+DigitalSignature+AES256+ DigitalSignatureWithTimeStamp+AES 256+DigitalSignatureWithTimeStamp	65	AES192+ DigitalSig nature
Trave lF+H otelD +Pick upH	AES128+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES256+DigitalSignature WithTimeStamp	65	AES128+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upB	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES256+DigitalSignature	66	AES192+ DigitalSig nature
Trave lF+H otelD +Pick upB	AES192+DigitalSignature+AES256+ DigitalSignatureWithTimeStamp+AES 256+DigitalSignatureWithTimeStamp	67	AES192+ DigitalSig nature
Trave lF+H otelD +Pick upB	AES128+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES256+DigitalSignature WithTimeStamp	67	AES128+ DigitalSig natureWit hTimeSta mp
Trave lF+H otelD +Pick upB	AES192+DigitalSignatureWithTimeSt amp+AES256+DigitalSignatureWithTi meStamp+AES128+DigitalSignature WithTimeStamp	67	AES128+ DigitalSig natureWit hTimeSta mp

The agent offers the composite service to the user by taking the best service set. The component services are executed as per the workflow. If any problem arises during execution of any of the component services, the system performs a rollback and starts the next best service set.

The best service set meeting user's requirement consists of the service providers TravelL for Travel service, HotelD for Hotel service and PickupK for Pickup service.

A comparison of the results obtained through the traditional method and the proposed method is given in Table 5.

Table 5: Comparison of Number of Service Sets Generated By Traditional Method and Proposed Method

Test Case	Total Number of Generated Service Sets by		% Improvement
	Traditional Method	Proposed Method	
Required service: Travel Hotel Pickup Security requirement: Confidentiality Yes Yes Yes Integrity No No No Authentication Yes Yes Yes Ready to pay the cost: Atmost 68 units	180000	30	99.983 %
Required service: Travel Hotel Pickup Security requirement: Confidentiality Yes Yes Yes Integrity No No No Authentication No No No Ready to pay the cost: Atmost 19 units	56644	30	99.947 %
Required service: Travel Hotel	10977120	30	99.999

Pickup Security requirement: Confidentiality Yes Yes Yes Integrity Yes Yes Yes Authentication Yes Yes Yes Ready to pay the cost: Atmost 64 units			%
Required service: Travel Hotel Pickup Security requirement: Confidentiality Yes Yes Yes Integrity Yes Yes Yes Authentication No No No Ready to pay the cost: Atmost 34 units	2274700	30	99.998 %
Required service: Travel Hotel Pickup Security requirement: Confidentiality No Yes Yes Integrity Yes No Yes Authentication Yes Yes No Ready to pay the cost: Atmost 52 units	460000	30	99.993 %

Table shows that the method proposed in this paper reduces the number of service sets to a very small number so that the users are presented with the most suitable service sets only.

6. CONCLUSION

For online businesses that offer complex services to users, QoS and security related issues are very important. The methodology proposed in this paper facilitates a business to offer the complex services that contain components that are compatible to each other with regard to QoS. The complex service formed also best meets the security requirements of the user. The method has been exhaustively tested. It is evident from the results that the presented methodology significantly increases overall efficiency of the system. The user is presented with very few service sets corresponding to most suitable composition that actually meet his requirements thus allowing him to use the most suitable composition. The method presented in the paper thus enables the business to offer the most appropriate complex services to the user.

In future, semantic approaches to determine compatibility based on QoS and security support can be investigated.

7. REFERENCES

[1] Pinar Senkul, "Composite Web Service Construction by Using a Logical Formalism", 22nd International Conference on Data Engineering Workshops (ICDEW'06), IEEE, 2006, pp 56.

[2] Wang Qing-Ming, Tang Yong, Zhang Zan-Bo, "Research In Enterprise Applications Of Dynamic Web Service Composition Methods and Models", Second International Symposium on Electronic Commerce and Security, IEEE, 2009, pp 146-150.

[3] Barbara Carminati, Chi Chi-Hung, Elena Ferrari, Lianghuan Yu, "Compatibility Driven and Adaptable Service Composition", Services Computing Conference APSCC2009, IEEE Asia Pacific, 2009, pp 396-401.

[4] Wang, Q.-M., Tang, Y., Zhang, Z.-B., "Research In Enterprise Applications of Dynamic Web Service Composition Methods and Models", Proceeding of Second International Symposium on Electronic Commerce and Security, IEEE, 2009, pp. 146–150.

[5] Boumhamdi, K., Jarir, Z., "Yet Another Approach for Dynamic Web Service Composition", Proceeding of International Conference on Internet Technology and Secured Transactions, IEEE, 2009, pp. 1–5.

[6] Eyhab Al-Masri, Qusay H. Mahmoud, "QoS-based Discovery and Ranking of Web Services", Computer and Communication Conference, ICCCN2007, IEEE, 2007, pp 529-534.

[7] Urjita Thakar, Nirmal Degdee, "An Approach to Discover Web Service Providers Based on Security Support", International Journal of Web Applications, Volume 1 Number 2, June 2009, pp. 47-56.

[8] Naiwen Lin, Ugur Kuter, Evren Sirin,"Web Service Composition with User Preferences", Proceedings of the 5th European Semantic Web Conference on the Semantic Web: Research and Applications, ESWC'08, 2008, pp. 629-643.

[9] Mikhail Matskin, Peep Küngas, Jinghai Rao, Jennifer Sampson, Sobah Abbas Petersen, "Enabling Web Services Composition with Software Agents", Proc. of Internet and Multimedia Systems, and Applications (IMSA 2005), Honolulu, Hawaii, USA, pp. 93-98.

[10] Fangfang Liu, Liang Zhang, Yuliang Shi, Lili Lin, Baile Shi, "Formal Analysis of Compatibility of Web Services via CCS", International Conference on Next Generation Web Services Practices (NWeSP'05), IEEE, 2005, pp. 6.

[11]Thirumaran, M., Dhavachelvan, P., Abarna, S., Aranganayagi, G.,"Architecture for Evaluating Web Service QoS Parameters Using Agents", Proceeding of International Journal of Computer Applications 10(4), 2010, pp. 15–21.