ABSTRACT
Web services (WS) are self-descriptive and platform independent applications that can be processed over internet. Enterprises nowadays has to rapidly suit the customer needs and improve business efficiency. Without a common set of standards, each organization is left to frame their own set of suitable business protocols. So, enterprises need heterogeneous services to be composed effectively. In this paper we are using decision making and capability assessment algorithm for predicting the composition process of the web services. This algorithm predicts the flaws and conflicts occurring in the composition process before execution. Since the conflicts are identified before process execution, this system reduces the time complexity and provides customer satisfaction. Decision making and capability assessment algorithm uses the parsing techniques to find the conflicts occurring in the composition. The proposed algorithm reduces response time and turnaround time. This improves user satisfaction and enhances degree of automation.

Keywords
Web Services (WS); Self-describing; Decision making and capability assessment; Heterogeneous service composition; Parsing Technique; Turnaround time.

1. INTRODUCTION
Web services facilitate executing pieces of functionality at remote locations. Examples of Web services are: Web search query, an address validity check, or a validation request for credit card data. More formally, the W3C defines a standard for Web service as “a software system created to support interoperable machine-to-machine interaction over the web”. Today, Web services are extensively used in the industry, and they have become essential for providing the countenance in the web application that is used every day. Many applications provide a Web service interface, which developers can use in order to coordinate several applications and to facilitate integrating an added-value feature from a third party.

Web services are proved to be a good choice for solving important dispute in the field of service integration. Before the emergence of Web services, developers had tough times to combine web service with another service running on a different operating system, using a different programming language, or running at a remote server. When developers wanted to use programmatic access to an application, e.g. for granting other applications to uses data, usually the technicians had to provide one implementation of the application and an application programming interface for the application per operating system, programming language, and so on. With Web services, only one application implementation with a standard-based API needs to be implemented and the client can use tools for generating code for a platform independent application. Because of its practical and easy usage, Web services are the central concept to develop platform independent applications.

However, the different service descriptions make the targeted consumer difficult to understand various types of services (e.g., SOAP-based services and RESTful services). With the everincreasing number of services proclaimed on the Internet (e.g., Google has recorded 230,000 Web Service description languages (WSDL) documents), finding needed services is just identical to looking for a needle in a haystack. A consumer cannot take the advantage to all the services leaving them to use mostly Web systems. A consumer cannot mix and match different services based on their conditions. Moreover, Web services facilitate limited support for an end user to integrate when no single service can relate the functions required by the consumer. An end-user has to go through several services on different URL addresses to integrate heterogeneous services. Imagine an situation such as planning a holiday which includes both buying a train ticket and booking a room for accommodation. An end user needs to decide on the time of the stay in a hotel based on the the availability and costs of train tickets and hotels. A user has to verify if a hotel is available before buying a train ticket and correspondingly check train tickets before booking a hotel. An end user has to shift between service providers to perform these necessary tasks. The situation becomes more complicated when an end user has to perform a task dependent on many different standards.

This proposed composition mechanism helps in identifying the conflicts occurring during the composition process. Since the conflicts were identified before starting the composition process, the turnaround time for the execution of the composed services reduces and user satisfaction rate is increased. Proposed model uses finite state machine to accurately capture the conflicts occurring in the composition process.

2. RELATED WORKS
In this section, a review of some heterogeneous service composition methods and summary of some disadvantages are addressed.

2.1. Heterogeneous Service Composition
Composing Heterogeneous services are required to improve the efficiency and to satisfy the customer needs. Heterogeneous services can be composed using different methods. In [9] the author explains how to compose heterogeneous services such as SOAP and REST. Since SOAP and REST are different in many aspects both services are needed to provide efficient service composition. In [10] the authors use adapters to compose the heterogeneous services and also compose mobile applications.
with the services. The SOAP supports WSDL file and the REST supports WADL file and the payload formats are also different for both the services. SOAP supports XML and the REST supports JSON data formats. Since both the services are different, we need to adapt the data format to the required format. The adapter was built by extending the BPEL engine which converts the REST data types to the SOAP service readable format.

REST is an architectural style and everything will be represented as resources. REST is quicker than the SOAP service in extracting the resources. But the security is lesser when compared with the SOAP services. So, we need to compose the SOAP and REST services to maintain the accuracy of the composition process. REST services use URI for processing the request and the response. The drawback of this composition process is that it does not identify the conflicts occurring during the composition process.

### 2.2. Model Driven Service Composition

The model driven service composition approach [12] helps in composing services dynamically. This approach provides a model through which the composition process will take place. The model was formed using the OCL. The author considers the business logics for the manipulation of the composition model. The model helps to user to perform the composition better than the existing composition system. BPEL engine provides the business logics to form the composition model. The model formed will give the details about the web services used for the composition and how those services were selected for the composition and who are the providers and how the heterogeneous binding between the services can be made. The disadvantage of this approach is there is no proper algorithm for computing the model.

### 2.3. Dynamic Service Composition

The dynamic service composition is the process of selecting the services for composition during the execution of the services [14]. Authors have proposed an automatic process which includes the verification and the validation of the services involved in the composition process. And the disadvantage in this approach is there is no mathematical model which identifies the errors and the exceptions accurately during the composition. The Context based Verification uses the context to identify the correct services and the composition will be carried out using the given context. This approach uses the WSDL for the composition process [16, 17]. The services which are adaptable to the given context will be considered for the composition process. It reduces the time complexity of the service composition and the processing time will be reduced using the context based model. Context based approach also supports runtime composition process. The above approaches consist of some disadvantages in the heterogeneous service composition process. The proposed approach solves the issues in the existing system and the composition will be effective and reduces the time complexity of the composition process.

### 2.4. Summary

Our proposed system solves the key issues in the existing system. The issues are: There is no mathematical model to identify the conflicts occurring in the composition process. There is no proper algorithm and FSM model incorporated to identify the conflicts accurately. Proposed system solves the above issues and makes the composition efficient.

### 3. PROPOSED WORK

We have proposed a prediction model which identifies the conflicts occurring during the heterogeneous service composition process. The conflicts are identified dynamically during the composition of the heterogeneous services.

#### 3.1 Decision making and Capability assessment Algorithm

This algorithm uses First and Follow functions to find and to predict the conflict occurring in the heterogeneous service composition process.

**Input:** Constraint Specification G.

**Output:** Decision table.

**Method:**

```plaintext
for (each production component → constraint in G) {
    for (each constraint in FIRST(component))
        add component → constraint to M[component, constraint];
    if (ε is in FIRST(component))
        for (each symbol constraint1 in FOLLOW(component))
            add component → constraint to M[component, constraint1];
    } make each undefined entry of M be error;

**Function** FIRST (component)

If entity is a component, then FIRST (component) = {component}

If component → ε is a production, then add ε to FIRST (component)

If component → constraint1 constraint2 ... constraint k is a production
    for k ≥ 1,
    and for some 1 ≤ k, constraint 1 constraint 2 ... constraint i derives the empty string, and a is in FIRST (constraint i), then add a to FIRST (component).

If constraint1 constraint2 ... constraint k derives the empty string, then add ε to FIRST (component).

**Function** FOLLOW (component)

Place $ in FOLLOW(S) where S is the start symbol.

If component → (constraint1) constraint2 (constraint3) is a production, then add every terminal in FIRST (constraint3) (except for ε if it is there) to FOLLOW (constraint2).

If there is a production component → (constraint 1) constraint2, or a production component → (constraint 1) constraint 2 (constraint3), where FIRST (constraint3) contains ε, then put everything in FOLLOW (component) into FOLLOW (constraint2)
```

The above algorithm helps in predicting the conflicts occurring in the composition process. The FIRST method is used to parse and find the exceptions occurring and the FOLLOW method helps in predicting the type of conflict occurred in the heterogeneous service composition process. The constraints are verified in the First method. The input specifications will be given by the user and those specifications are converted into grammars. As we use finite state machine model in this approach, the conflicts are identified accurately. The services are represented as states and the transitions are done based on the specifications.
3.2 Working of Proposed Algorithm

Let us consider the SOAP message of bank application. The SOAP message will be converted into the specifications and those specifications are fed into the algorithm and the conflicts will be predicted using this algorithm.

**FIRST:**

First(DTME) = First(Soap:env) = {<xmlns:soapenv>}
First(H) = {<soap:header>}
First(B) = First(create_acc) = First(acc_no) = {<tem:accno>}
First(V1) = {(a|b|c..)+}
First(uname) = {<tem:uname>}
First(V2) = {(1|2|…)+}
First(amt) = {<tem:amt>}

**FOLLOW:**

Follow(soap:env) = Follow(B) = {$}
Follow(H) = First(B)+Follow(B) = {<tem:accno>,$}
Follow(acc_no) = First(uname)+Follow(uname) = {<tem:uname>,<tem:amt>,$}
Follow(uname) = First(amt)+Follow(amt) = {<tem:amt>,$}
Follow(V1) = Follow(acc_no) = {<tem:uname>,<tem:amt>,$}
Follow(V2) = Follow(uname) = {<tem:amt>,$}

The above results are split into three categories for identification purposes. They are: Conflicts, Exceptions and Errors. The conflict arises when the parser finds the incompatible types of specifications. Exceptions occur when there is a problem in the flow of execution of the process. And the errors occur when the state or condition becomes wrong. This predictive algorithm helps in identifying the exceptions, errors and conflicts occurring during the composition process.

<table>
<thead>
<tr>
<th>EXCEPTION</th>
<th>NAME OF THE EXCEPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ex1</td>
<td>Invalid Tag Exception</td>
</tr>
<tr>
<td>Ex2</td>
<td>Invalid Argument Exception</td>
</tr>
</tbody>
</table>

Table 1. Exception table

<table>
<thead>
<tr>
<th>ERROR</th>
<th>NAME OF THE ERROR</th>
</tr>
</thead>
<tbody>
<tr>
<td>Er1</td>
<td>Type Mismatch Error</td>
</tr>
</tbody>
</table>

Table 2. Error table

<table>
<thead>
<tr>
<th>CONFLICT</th>
<th>NAME OF THE CONFLICT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cl2</td>
<td>Illegal argument conflict</td>
</tr>
<tr>
<td>Cl3</td>
<td>Illegal state conflict</td>
</tr>
</tbody>
</table>

Table 3. Conflict table

The above table shows the identified errors, exceptions and conflicts. The conflict occurs when the tag appears in the place of terminals and vice versa. Exceptions occur when the non-terminals replace the terminals. Errors occur when the condition fails. In this concept, state error occurs when the data type mismatch occurs during the parsing.

Though the conflicts, errors and exceptions occur during the composition of SOAP and REST services, it is necessary to compose the services to manage the demands of the customers. The above algorithm uses FIRST and FOLLOW methods to identify the conflicts occurring during the heterogeneous composition process. This identification process helps in reducing overall turnaround time. Decision making and capability assessment algorithm predicts the conflicts and intimates the user. The heterogeneous composition process helps in reducing the execution time and increases the efficiency of the system. The below table shows the identified errors, exceptions and the conflicts occurred during parsing process.

The SOAP request message will be taken and the request message will be converted into the specifications. The generated specifications are parsed using the decision making and capability assessment algorithm. This algorithm checks whether the given SOAP message satisfies the given constraints or not. If the SOAP message not satisfy the constraint, then the algorithm identifies whether it is a conflict or error or exception and intimates the user about the condition and suggest the user to select the conflict free service composition.

The composition flow suggested by this algorithm reduces the turnaround time of the heterogeneous composition and gives preferences for the user’s opinion during heterogeneous composition. Decision making and capability assessment algorithm addresses the capabilities of the services. Then the services with the required capability will be suggested to the user.

Table 4 describes how accurately we predict the conflicts and the exceptions occurring in the composition flow. The prediction table shows the identified errors for the above example.
Table 4 PREDICTION TABLE
<table>
<thead>
<tr>
<th>STACK</th>
<th>INPUT</th>
<th>OUTPUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[soap:env]$</td>
<td>$<a href="">wsse:timestamp</a><a href="">xmlns:soapenv</a><a href="">soapenv:header</a>...<a href="">tem:accno</a>(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B][H]<a href="">xmlns:soapenv</a>$</td>
<td>$<a href="">soapenv:header</a>...<a href="">tem:accno</a>(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B]<a href="">soap:Header</a>$</td>
<td>$<a href="">tem:accno</a>(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B][create_acc]$</td>
<td>$<a href="">tem:accno</a>(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B][amt][uname][accno]$</td>
<td>$<a href="">tem:accno</a>(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B][amt][uname][V1]<a href="">tem:accno</a>$</td>
<td>$<a href="">tem:accno</a>(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B][amt][uname][V1]$</td>
<td>$(1</td>
<td>2</td>
</tr>
<tr>
<td>$[B][amt][uname][V1][1</td>
<td>2</td>
<td>3..]$</td>
</tr>
<tr>
<td>$[B][amt][uname]$</td>
<td>$(a</td>
<td>b</td>
</tr>
<tr>
<td>$[B][amt][V1]<a href="">tem:uname</a>$</td>
<td>$(a</td>
<td>b</td>
</tr>
<tr>
<td>$[B][amt][V2]$</td>
<td>$(a</td>
<td>b</td>
</tr>
<tr>
<td>$[B][amt][V2]<a href="">tem:uname</a>$</td>
<td>$(a</td>
<td>b</td>
</tr>
<tr>
<td>$[B][amt]$</td>
<td>$<a href="">tem:amt</a>$</td>
<td></td>
</tr>
<tr>
<td>$[B]<a href="">tem:amt</a>$</td>
<td>$<a href="">tem:amt</a>$</td>
<td></td>
</tr>
<tr>
<td>$[B]$</td>
<td>$&lt;$</td>
<td>$&gt;$</td>
</tr>
</tbody>
</table>

Table 5 PARSING TABLE
4. EVALUATION METRICS

To evaluate the proposed system following metrics are used.

Precision and recall are calculated with the help of the webservice information. Precision is the important measure for identifying the accuracy of the services. It is defined as the ratio between the numbers of web services satisfying the given constraint to the total number of the relevant web services.

\[
P = \frac{W(x)}{p(x)}
\]

Where,
- \(W(x)\) is the number of identified relevant services.
- \(p(x)\) is the total number of relevant services.

Recall is the fraction of related services that are repossessed. Recall is formulated by the ratio of Number of services satisfying the constraints to the total number of services in the repository. The formula for recall is given by,

\[
R = \frac{W(x)}{V}
\]

Where,
- \(W(x)\) is the number of identified relevant services.
- \(V\) is the total number of services in repository.

Response time is the percentage of how effectively the grammars are generated and the parsing of those grammars from the given set of specifications. The formula for response time is given by,

\[
R_{sc} = \left(\frac{H}{V}\right) \times 100
\]

Availability denoted by \(A\) is the percentage of webservics that are available to the system while processing the user’s request. It is calculated with the use of a number of successful services and the number of failure service. The formula for it is given by,

\[
A = \left(\frac{U_s}{U_s + D_s}\right) \times 100\%
\]

Turnaround time is the total time taken between the submission of a request for execution and the return of the complete output to the customer/user. It may vary for various programming languages depending on the developer of the software or the program. Turnaround time may simply deal with the total time it takes for a program to provide the required output to the user after the program is started.

\[
T = T_{request} + T_{response} + T_{execution}
\]

5. EXPERIMENTAL RESULTS

In this section, we use NetBeans IDE to simulate the results. The java class Syntax analyzer contains functions to get the grammar input, split the terminals and Non-terminals, and find the First and Follow sets for given specification and to predict the errors. The screenshots of the results are shown below.

5.1 SAMPLE INPUT AND OUTPUT

1. Grammar Input

The above screenshot represents the input specification of the SOAP service to test using the Decision making and Capability Assessment algorithm.

2. Terminals and Non-Terminals
The screenshot 2 shows the splitting of Terminals and Non Terminals from the given specification to evaluate the capability of the service.

3. Capability Assessment

The capability of the service will be assessed using the decision making and capability assessment algorithm explained in section 3.

4. Prediction Table

The screenshot 4 describes the prediction table generation for the service specification. The e0, e1, e2 in the screenshot represents the errors and c0, c1 and c2 represents the conflicts.

5.2 METRICS

Precision and Recall for the existing and proposed system are represented by the graphs illustrated in Figure 10. Commencing the graphs, it is determined that the precision for the existing is lesser when equated to the precision of the proposed, this point toward that the dynamic recommended services and REST recommended services bring into being useful for a user was more sophisticated than the existing set of static services.
6. CONCLUSION

Heterogeneous service composition is an emerging technique to satisfy the customer needs. This composition process provides many conflicts during the execution process. The Decision Making and Capability Assessment algorithm helps to identify the conflicts occurring during the heterogeneous composition process. The finite state machine model was incorporated in this algorithm to identify the conflicts accurately. The prediction table identifies the conflicts, exceptions, and errors occurring during the composition process. And the parsing table helps in identifying those conflicts efficiently. The experimental result shows that this identification reduces the turnaround time of the composition since the conflicts are identified before the composition process. This algorithm helps the user to select the appropriate service for composition by suggesting the composition flow to the user.

REFERENCES


