ISSN 2047-3338

Adaptive Web Service Composition Based on Interface Description

Y. Oussalah[1] and N. Zeghib[2]

[1,2]LIRE Laboratory, Computer Science Department, University of Mentouri, Algeria

***Abstract*— Web services have received much interest due to their potential to design and build complex inter-enterprise business applications. A particular interest concerns dynamic Web services composition that offers the opportunity for creating new Web services at runtime from those already published. In this paper we focus on mismatches occurring during dynamic composition of Web services. These mismatches require adaptation to insure the correct working of the involved components in the service composition. We propose an approach for dynamic and automatic composition and adaptation of Web services. The approach is based on the information that is already available in interface descriptions. The approach allows programmers to define dynamic Web service composition and adaptation without changing the source code.**

***Index Terms*— Web Service, Dynamic Composition, Interface Description and Adaptation**

I. INTRODUCTION

Services Oriented Architecture (SOA) uses the concept of service as an elementary brick to assemble complex systems. It provides means for the self description, announcement, discovery, interaction and usage of services. Nowadays, an increasing amount of companies and organizations only implement their core business and outsource other application services over the Internet. Thus, the ability to efficiently and effectively select and integrate inter-organizational and heterogeneous services on the Web at runtime is an important step towards the development of the Web service applications. Unfortunately, individual Web services cannot satisfy all the service requests. When that happens, it is desired to seek possibilities of combining existing services together to fulfill the request. Particularly, the dynamic web service composition is very promoting because it enables the user to select, at runtime, existing web services to provide an unlimited number of new services from limited set of services. This dynamic feature of service composition provides flexibility and adaptability to applications. For example, an application built on top of the dynamic service composition system is able to change its user interface dynamically according to user's preference (e.g., English/Japanese menu, colorful/simple buttons, …etc.), Furthermore, a totally new application may emerge by combining several components designed for entirely different purposes.

In the research related to Web services, several initiatives have been conducted with the intention to provide platforms and languages that will allow easy collaboration, composition and integration of heterogeneous systems. In particular, some standards have been developed such as Universal Description, Discovery, and Integration (UDDI) [1], Web Services Description Language (WSDL) [2], Simple Object Access Protocol (SOAP) [3], Business Process Execution Language for Web Service (BPEL4WS) [4].

Despite all these efforts, the Web service composition still is a highly complex task. One source of this complexity is the mismatches that may occur between two services in the composition process. In fact, this may appear at different levels: signature, behavior, quality of service and semantics. Hence, there is a need for adaptation method to correct these mismatches without modifying the service code due to its black-box nature. The adaptation ensures correct working and interaction among the involved components in the composition.

In this paper we present an efficient algorithm to support dynamic composition and adaptation of Web services. Especially, we use the interface descriptions of services to detect the mismatches between interfaces and we perform the recovering of structural and behavioral mismatches via a set of mapping operations.

The rest of this paper is organized as follows. Section II introduces basic concepts of web service composition, compatibility and adaptation. Section III describes service interface. Section IV highlights the mismatch scenarios and gives a motivating example. Section V presents the proposed approach. Section VI discusses related work and existing approaches supporting Web service adaptation and composition. Finally, last section concludes the paper with future works.

II. BASIC CONCEPTS

Prior to the presentation of the proposed approach, we introduce basic concepts related to web services composition, compatibility and adaptation.

A. Web Service Composition

Given the current proliferation of Web services, service composition appears as an important strategy to implement distributed applications. For example, if French to Chinese translator does not exist, but there are French to English and English to Chinese translators, each one implemented in a Web service, the French to Chinese translator may be created through their composition.

In practice, the Web services composition can be done in a static or dynamic way. The static composition allows the requestor to create an abstract model that should be respected during the execution of these Web services. While the dynamic composition enables selecting the atomic Web services automatically and combines them to create an unlimited number of new Web services. The dynamic composition is very challenging as it is done at runtime based on the user's request. With dynamic composition, an unlimited number of new services can be created from a limited set of service components. Dynamic composition is more suitable if the process has to adapt dynamically to unpredictable changes in the environment. However, the dynamic web service composition may lead to several faults such as poor response, incorrect order, service incompatibility, and unavailability. If the failure occurs, the cause of the failure has to be detected and healed. Since the web services composition is done dynamically, the services need to reconfigure themselves when the environment changes without any human intervention and without stopping the composite service [5].

B. Compatibility in the Web Service Composition

To ensure the correct working between the involved components in the composition there should be compatibility between them i.e. they can invoke each other and the result of the composition can be issued. In fact, the compatibility may be affected when some heterogeneities occur between services:

1) The provided messages are delivered as flow when they are required as single message.
2) The provided message is delivered as single when it is required as flow of messages.
3) The provided messages contain irrelevant messages or additional parameters so the additional parameters should be hidden.
4) The type of the provided message does not match the type of the required one.

The compatibility in the composition concerns not only the exchanged messages but also the correct sequence of ordered operations which can be achieved by combining compensable operations.

Consequently, the compatibility between services can be seen from two perspectives: structural (where the focus is on the messages types) and behavioral (where the focus is on control dependencies between message exchanges).

C. Adaptation

Due to the dissemination of ubiquitous and autonomic computing, several issues related to adaptation have been widely studied. In the context of Web services, adaptation comes from the fact that services may be reused in context for which they were not originally designed. Thus service reuse leads to situations where a service is needed to participate in multiple collaborations where various interfaces are required from it. This requires adaptation of provided interfaces to the required ones. This mechanism is known as *web service interface adaptation*.

Adaptations can be static or dynamic, and manual or automatic [6]. The static adaptation is carried out through modifications in the source code, while the dynamic one modifies software runtime characteristics. Manual adaptation means direct intervention in the system, whilst automatic one can be performed by the system itself.

III. WEB SERVICE INTERFACE

Web Services are autonomous software components that can be published, discovered and invoked for remote use. For this purpose, their characteristics must be made publicly available under the form of Web service descriptions. The business world has developed a number of XML-based standards to formalize the description of Web services. WSDL is the current standard of Web service description. Web services are considered as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate [2].

Much functionality can be contained in one Web service, and each is implemented by an operation. A Web service can be expressed as a set of operations. An operation is specified by its name, its input and output message types, i.e. *o: =<name, data Input, data Output >*, which is the interface of the Web service.

The WSDL interface document defines the message format for operations and messages defined by a particular portType. We can generate a monolithic WSDL document that contains all WSDL elements, or a separate WSDL interface document. A <message> element is needed to compose such data types into messages. Messages need to be grouped into operations, which may define an <input>, an <output> and a <fault> message. Here is the simplified structure of WSDL [7]. Figure 1 shows the simplified structure of the WSDL description.

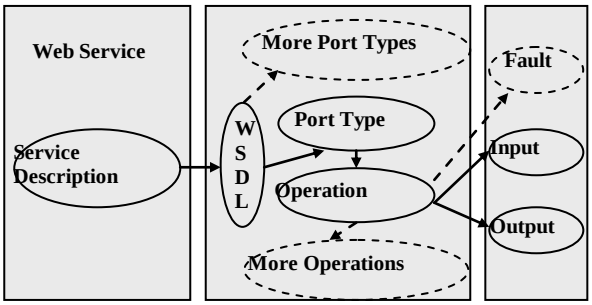

Fig.1. Simplified WSDL structure

IV. MOTIVATING EXAMPLE

The scenarios in the Figure 2 show possible mismatches which may occur at runtime between the involved components in the composition. In the first scenario *(a)* the Web service sends two different messages (*A* and *B*) while only one of them (*A*) is expected. In the scenario *(b)* messages are sent aggregated (*A*+*B*) when they are needed to be separated. The third scenario (*c)* is the reverse of *(b)*: the messages are sent separately when they are needed in aggregation. In the last scenario (*d)*, the type of the sent message does not match with the required type.

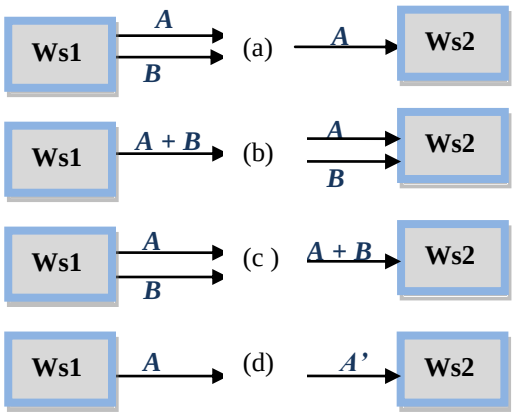

Fig. 2. Mismatch scenarios

In the aim to illustrate these mismatches, let's consider the example of getting the weather report from an *Ip address*. We suppose that the existing Web services do not perform the required task, whereas the composition of the Web services **ResolveIp** and **GetWeather** may be suitable : the first Web service can provide the location corresponding to a given *Ip address*, while the second gives the weather for each location. The input and output of both Web services are illustrated in Table I.

Table I. Input and Output of **ResolveIp and GetWeather**

Web services	Input	Output
ResolveIp	*Ip*:String *License* :String	*City*:String *StateProvince* :String *Country*:String *Latitude*:string *Longitude*:String *CountryCode*:String *Region* name:string
GetWeather	*CityName*:String CountryName:String	GetWeatherResult : String

Such mismatch is easily highlighted using JOpera plugin [8], as shown in Figure 3. The process **Weatherfromip** cannot provide the value of weather (step 5) due to the mismatch occurring at step 4.

The below scenario may be summarized in the following steps:

0) The process of composition of the two web services: **ResolveIp** and **GetWeather**.
1) The input parameters are: *Ip* and *license*.
2) The passage of parameters to the operation ResolveIP.
3) The output of ResolveIp is a complex type (as mentioned in the Table 1).
4) Mismatch: the output of ResolveIp does not satisfy the input of GetWeather.
5) No value returned because the operation GetWeather cannot produce the output.

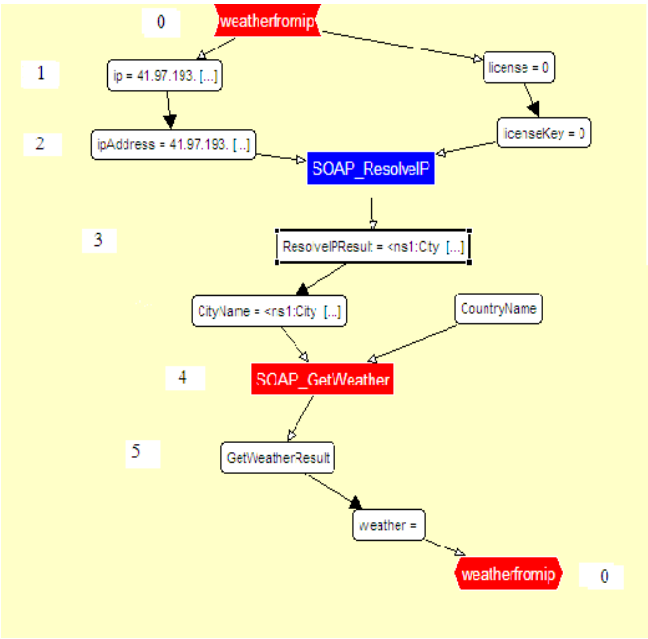

Fig. 3. Mismatch scenario in **ResolveIp** and **GetWeather** composition

V. ADAPTIVE WEB SERVICE COMPOSITION

Interactions between Web services involve the exchange of messages. A message consists of one or more parameters, each having a data type. Hence it is important to check if the data types and number of the parameters sent by a service are compatible with the parameters required by its partner. This requires pre-conditions of input and post-conditions of the outputs. Thus composing two Web services require finding two compensable operations(one of each) that can be linked: two operations can be linked when the output parameters of the first (source) can cover the input parameters of the second (target).

Automatic services composition relies on the automatic matching of inputs/outputs of operations in Web services, i.e. interface matching. In our approach we propose an automatic and dynamic composition based on user request to choose adequate services to perform composition and adaptation when this latter is needed. To achieve our goal, we introduce transformation operations of interfaces [9] which will be used in the process of interface adaptation. Mainly we consider in the algorithm the four following operations:

- Collapse: is used when a stream of messages is aggregated into a single message.
- Burst: works in the reverse of the Collapse operation and it is used when a single message needs to be split into a stream of messages.
- Hide: is used when a message from the source interface is not required in the target interface.
- resvType: is used when the type of message provided in the source interface is not compatible with the required one in the target interface.

A. *Main Steps of the Approach*

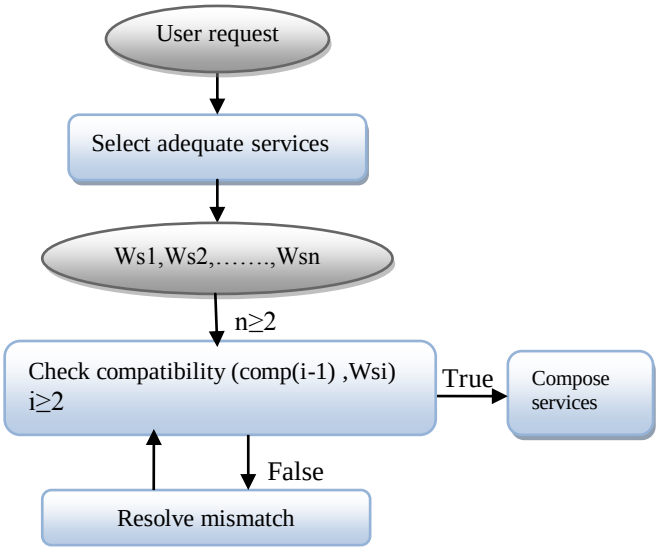

Fig. 4. Main Steps of the web service composition and adaptation

The proposed algorithm performs sequential composition of two Web services. For more than two services, the function of composition may be applied recursively. The algorithm requires the ability to discover or detect pairs of services such that the output of one service is equal or equivalent to the input of another (correspondence between interfaces). If the mismatch occurs, the adaptation will be performed using the mapping operations presented above. Hence, we resolve inadequacy resulted due to number and type of parameters.

The selection of adequate services for composition will be according to the outputs and inputs messages of operations if they satisfy the input and the output of the user request, starting from a known input, in order to compute the desired output.

For the sake of simplicity we suppose that when selecting the adequate services for composition, the function of selection will reorder the services according to their compensable operations which means : finding compensable operations that can be linked. Hence, the operation of selecting the adequate services performs a sequence planning from a given input to produce the desired output. This insures behavioral compatibility which concerns the dependences between messages.

The composition of n Web services is defined recursively by the function Rec_comp as follows:

Rec_comp (n) = compose (Rec_comp (n-1), Wsn) if n ≥ 2 .

Rec_comp (1) = Ws1.

For instance:

Rec_comp (2) = compose (Rec_comp (1), Ws2) = compose (ws1, ws2).

B. *The Algorithm*

Let's consider the service interface of a web service

I = (input,SO,output) , SO = set of operations.

Begin : (input = user_request)

Declaration

```
Boolean compatibility = false ;
Webservicelist wsl ={} ;
Main() {
 wsl= search_for_adequateservcies(user_request);
 If ( wsl.length == 1) then // no need for composition //
      Invoke(wsl(0));
If( wsl.length >=2) then // wsi are the found services//
 For (i = 2 to n) do
       {
   compatibility = Compatibility_checking(comp(i-1),wsl(i)) ;
     while (compatibility == false ) do {
       Resolve_mismatch() ;
    compatibility =Compatibility_checking(comp(i-1),wsl(i)) ;
     }
   Comp(i) = compose_Services(comp(i-1),wsl(i));
                  }
  }
```

Compatibility_checking(service A, service B) *{*
If (output(A) .Satisfy(input(B)) &
(SO(A) . its related _Operation(SO(B)))
then compatibility = true ; }

Resolve_mismatch() *{*
If (outputA :: List & inputB :: Single) then
// cardinality of *List* is greater than 1//
collapse (outputA) ;
If (outputA :: Single & inputB :: List) then
Burst (outputA) ;
If (output1.hasAdditionalPar() = true) then
// hasAdditionalPar: has additional parameter//
hide (outputA,index) ;
// the index is used to specify the parameter to hide //
If NOT(outputA.typecompatible(InputB.type) then
resvType(outputA.type)
}
End.

C. Illustrating Example

Let's illustrate the use of the algorithm through an example. We consider the example shown in the Table which presents list of selected Web services as a response for the user request "getWeatherReport".

We apply our algorithm to perform the composition. It's known that the constraint of sequential Web service composition is that the output the of the former service should satisfy the input of the next one. The possible combinations as response for such service request are: (ws1,ws2) or (ws1,ws3,ws4) or (ws1,ws5,ws4).

Table II. The selected web services for composition

Ws	Name	operation	Input/type	Output/type
ws1	getGeoIp	getgeoIp	IP:String	City: String
ws2	getWeather	getWeather	City:String	Weather -result : String
ws3	MediCare-Supplier	GetSupplier-ByCity	City:string	Zip: String
ws4	USWeather	GetWeather-Report	ZipCode:Int	Weather -Report: String
ws5	ZipcodeLook-upService	CityTo -LatLong	city:string	Zip:String

The composite service (ws1,ws2) is selected to be the best combination of services to satisfy the user request. In this case the invoked service is comp(ws1,ws2) which results the weather report as requested by the user .

If we choose the combination of services (ws1,ws3,ws4), the heterogeneity will occur when comparing the output3 and input4: they have different data types (output3.type=String, input4.type= Integer) and this mismatch will be resolved by mapping the type of output3 to the type of input4 to insure the service communication.

We have developed a tool ***CompAdapt***[1] that implements the presented algorithm in Java.

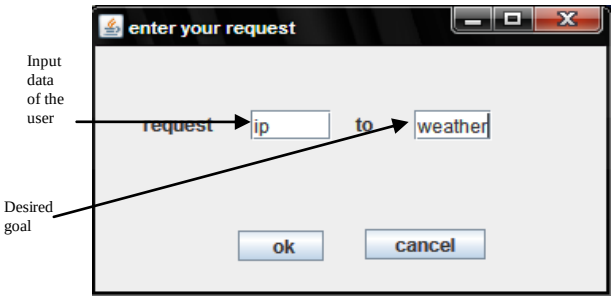

Fig. 5. ***ComAdapt*** 's user interface(for entering the user request)

The ComAdapt *tool* uses a database to store Web service descriptions (as UDDI register). It retrieves the description of the desired Web services to perform the composition as shown in Table III.

Table III. Database of Web services descriptions

webservice	input	typeIn	nameOP	output	typeOut
comp(ws1,ws2	ip	String	getgeoIp,getWeath	weather	String
ws1	ip	String	getgeoIp	city	String
ws2	city	String	getWeather	weather	String
ws3	city	String	GetSupplierByCity	zip	String
ws4	ZipCode	int	GetWeatherReport	Weather	String
ws5	city	String	CityToLatLong	zip	String

Fig. 6. The composite service with its input and output

[1] Composition Adaptation

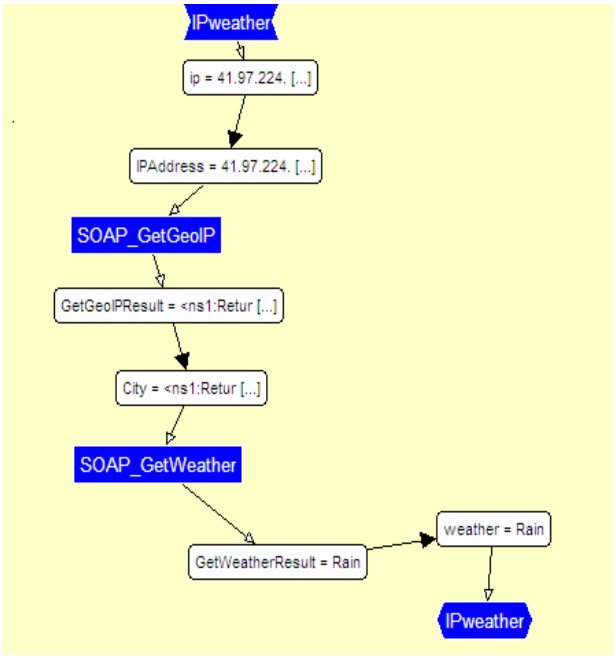

Fig. 7. Web service composition using JOpera (weather result from Ip)

VI. RELATED WORK

Service interfaces can be described from a structural perspective (where the focus is on message types) and from a behavioral perspective (where the focus is on control dependencies between message exchanges). The problem of interface adaptation from the structural perspective has received considerable attention leading to a number of transformation definitions such as XSLT [10] and schema mapping tools such as Microsoft BizTalk Mapper [11], Stylus Studio XML Mapping Tools [12], and SAP XI Mapping Editor. However the problem of interface adaptation from behavioral perspective is still open. A number of studies in this field have been proposed. For instance, in [9] the authors describe the interface as ordered sequence of actions and they have proposed algebra of transformation of interfaces, depending on the cases of mismatch that could occur to resolve inadequacy between interfaces. They take as input a source interface to produce a target interface by transforming the interfaces via six operators which are:

Flow: where a defined action in the source interface becomes another action in the target interface.

Gather: is applied when multiple actions from the source interface map to a single action in the target interface.

Scatter: is applied when a single action in the source interface is transformed into multiple actions in the target interface.

Collapse: is used when a stream of messages resulting from multiple instances of the same communication action is aggregated into a single message.

Burst: works in the reverse of the Collapse operator and is used when a single message needs to be split into a stream of messages.

Hide: is used when an action from the source interface is not required in the target interface.

In [13] authors proposed an approach to the composition and adaptation of mismatching components in systems where the number of transactions is not known in advance. Their approach applies composition at run-time with respect to the composition specification, using π-calculus to specify component interfaces.

In [14] authors specify mediator with finite state automata that resolves behavioral mismatches at runtime due to the removal of operations in provided interfaces, they also proposed an algorithm that resolves such mismatches .

In [7] authors have proposed an approach for composition that only uses already available information in service interface definitions. It does not require service providers to describe their interfaces with semantic markup. They proposed data types matching and service composition algorithm, using the measure of linguistic similarity between two data types.

In [15] authors present a framework for Dynamic service composition and parameters matchmaking. They discussed main problems faced by dynamic service composition. Among which are transactional support and compositional correctness. To make the system flexible they include user involvement at few steps for example selection of service and matchmaking decision.

In [16] authors propose a process mediation architecture based on Triple space computing, and present potential solutions for resolvable message sequence mismatches. In addition, they categorize these resolvable mismatch scenarios into five classes. This analysis generalizes the resolvable message sequence mismatches, provides the basis for checking Web service compatibility from the behavioral aspect, and offers an opportunity to have a uniform solution to address these mismatches.

In [17] authors have identified number of possible mismatches between services and some basic mapping functions that can be used to solve simple mismatches. Such mapping functions can be combined in a script to solve complex mismatches. Scripts can be executed by a mediator that receives an operation request, parses it, and eventually performs the needed adaptations.

In our approach we present an algorithm that support both dynamic composition and adaptation. Our contribution regarding the most approaches is that we have used the dynamic composition and adaptation whereas the other approaches resolve either the dynamicity of the composition or the dynamic adaptation of static composition.

VII. CONCLUSION

We have presented an algorithm performing automatic and dynamic composition of web services. The searching of services is based on user request at runtime. The algorithm also performs adaptation (if the heterogeneity occurs between service interfaces). This adaptation is performed via mapping operations to fulfill the compatibility requirement. Thus, the services may interact and be interchangeable at runtime. Consequently, our algorithm enhances the flexibility of the interfaces to insure the correct working and interaction among them. We have also implemented the algorithm in Java language to specify the process of web service composition and adaptation (ComAdapt tool), and to prove the efficiency of our approach.

In future it will be interesting to perform adaptation in more general cases such as when the composition includes different partners producing the output from different services that should satisfy their next input (including parallel composition). In this case, more operations might be needed to perform adaptation and to insure the correct working among services.

REFERENCES

[1] T. Bellwood and al. "Universal Description, Discovery and Integration specification (UDDI)", http://uddi.org/pubs/uddi-v3.00-published-20020719.htm., 2002.

[2] R. Chinnici and al. "Web Services Description Language (WSDL) ", http://www.w3.org/TR/wsdl/, 2001.

[3] D. Box and al. "Simple Object Access Protocol (SOAP)" , http://www.w3.org/TR/SOAP/, 2001.

[4] T. Andrews et al. "Business Process Execution Language for WebServices(BPEL4WS)",http://www106.ibm.com/developerworks/webservices/library/ws-bpel, May 2003.

[5] S.Poonguzhali, R.Sunitha, and G.Aghila,Self-Healing in Dynamic Web Service Composition, International Journal on Computer Science and Engineering, Vol. 3, No. 5. (2011), pp. 2054-2060.

[6] Lins FAA, dos Santos JC and Rosa NS "Improving Transparent Adaptability in Web Service Composition", Proc. IEEE International Conference on Service-Oriented Computing and Applications, Newport Beach, 2007,CA, pp.80–87 .

[7] J. Zhang, S.Yu, X.Ge and G.Wu, "Automatic Web Service Composition Based on Service Interface Description", In Proceedings of International Conference on Internet Computing'2006. pp.120-126.

[8] http://www.jopera.org/

[9] M. Dumas, M. Spork and K. Wang, "Adapt or Perish: Algebra and Visual Notation for Service Interface Adaptation", 4th International Conference on Business Process Management, 5-7 September 2006, Vienna, Austria.

[10] http://www.w3.org/TR/xslt

[11] http://msdn.microsoft.com/enus/library/ee253382(v=bts.10).asp

[12] http://www.stylusstudio.com/xml_schema.html

[13] J. Camara, G. Salaun, and C. Canal, "Run-time Composition and Adaptation of Mismatching Behavioural Transactions", Fifth IEEE International Conference on Software Engineering and Formal Methods SEFM 2007, London, pp. 381-390.

[14] Aït-Bachir and M. Fauvet , "Reconciling Web Service Failing Interactions: Towards an Approach Based on Automatic Generation of Mediators", 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, WETICE 2007, Paris.

[15] M. Allauddin, F. Azam, Dynamic Web Service Composition and Parameters Matchmaking, International Journal of Computer Applications 36(9):21-26, December 2011. Published by Foundation of Computer Science, New York, USA.

[16] Z. Zhou, B. Sapkota, E. Cimpian, D. Foxvog, L. Vasiliu, M. Hauswirth and P. Yu: "Process Mediation Based on Triple Space Computing". Proceedings of the 10th Asia-Pacific Web Conference, April 2008, Shenyang, China.

[17] L. Cavallaro and E. Di Nitto, "An Approach to Adapt Service Requests to Actual Service Interfaces", In: SEAMS '08 Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. ACM, New York, NY, USA, pages 129–136.