An Approach to Support Monitoring and Recovery of BPEL Processes at Runtime

Kaouthar Boumhamdi
Computer Science Department
Faculty of Sciences, Cadi Ayyad University
Marrakesh, Morocco

Zahi Jarir
Computer Science Department
Faculty of Sciences, Cadi Ayyad University
Marrakesh, Morocco

ABSTRACT
The dynamic composition of Web service can be modeled as a service oriented workflow. However during the execution of this workflow, potential failure can occur on-the-fly which make down the composed Web service. To avoid downtime of the overall process in execution, some recovery methods are required. The objective of this paper is to present a state of the art of the most known failures and the more important recovery methods that can be used to recover the execution workflow in failure. Also, it proposes an implementation of our proposed architecture to ensure a better QoS of the composed Web service. This architecture has the privilege to detect and recover dynamically failure according to the user's requirements and/or the availability of main critical resources.

General Terms
Web Service Composition, Recovery methods, Composition failures, Monitoring.

Keywords
Dynamic Web Service Composition, Composition failures, Recovery methods, Monitoring, BPEL.

1. INTRODUCTION
The ability to compose several existing Web services in order to satisfy certain requirements and to avoid downtime of the overall process execution if a failure occur during the execution, is still an important challenge in SOA paradigm. This challenge becomes more complex due to the dynamic nature of the execution environment.

Several types of failure can be fired when composition plans are in execution. We cite for example the case when servers' providers are busy, or a broken connection is happened. In this case, timeout detector may be used to consider this kind of failure. However, other kinds of failure are difficult to be detected and may cause big problems if not caught in time. For instance, we present the situation when a service provider updates their services by changing the types of the expected input/output data which can be done periodically. Consequently, client using these services will receive corrupt data from the service. Thereafter this type of failure will cause incorrect composition workflow due to the incorrect call of composed Web services.

Due to failures that can occur during the execution of composed web services, various methods have been proposed, but unfortunately the challenge is still evoked. These methods include transactional methods, Self-healing networks or QoS constraints as a heuristic to recover dynamically composition of services, etc.

In this paper, we try to classify the most common failures that can occur when using Web Services, to discuss failure detection strategies used and to detail the most known recovery methods.

The remainder of the paper is structured as follows: Section 2 presents a background of failure types, failure strategies detection and recovery methods. Section 3 summarizes some important related research studies that give a solution that fulfill a dynamic composition of web services, failure detection and recovery issues. Section 4 details an implementation for our proposed solution called FlexCoM4WS that provides a Flexible Composition through Monitoring Environment for Web services. Finally, Section 5 presents a conclusion and some perspectives.

2. BACKGROUND FAILURES AND RECOVERY
The objective of this section is dedicated to failures and recovery methods description in context of web service composition. After defining generally which a failure is, we present the more identified categories and types of it and also expose the proposed recovery methods in the literature that can be handled if a failure is occurred.

2.1 Categories of failures
The failure can be defined as an abnormal condition or defect at the component, equipment, or sub-system level which may lead to hazardous consequences. To ensure a better quality of the composite service we have to cope with different error messages and the failures that can happened during the dynamic composition execution. It is therefore important that the faults that cause these failures can be classified to apply the correct failure recovery method for that failure. Motallebi in [1] has classified failures into three main categories: physical, development, interaction as faults follow:

- Physical faults are observed as failures happened in the network, or on the server provider side.

- Development faults may be introduced into a system by its environment. For instance, we find parameter incompatibility fault, workflow inconsistency, or non-deterministic action fault.

- Interaction fault occurs when the given composite service fails more frequently or more severely than acceptable such as incorrect order, execution fault, response fault, and incorrect service fault.

2.2 Types of failures
There are different types of failures which can occur during the use of web services. Steyn in [2] has enumerated the more

frequent failures that might manifest during the execution of a web service such as a broken connection, busy or downtime server, etc. He also presents and classifies the most happened of them as follow:

- Failures caused by availability that can be due to the state of the server or the connection to a server. These kinds of failures can be present in the form of a 'time out', or a 'service not found' error.
- Failures caused by concurrency that come all from the usual concurrency problems. In this case a web service can be used by more than one client at any time, and this can cause problems if the service is being updated by a client side or by a server side.
- Failures caused by dependency when services can make use of other external services to gather the required information before passing it on to the client.
- Failures caused by incorrect composition that may also happen during the composition phase.
- Failures due to ambiguous output.
- Partial failures caused by incorrect parallel execution: A partial failure implies that during a parallel execution of services, one of the branches cannot find the needed or requested services. This implies working with incomplete data.

After exposing a brief background of failures, the next subsection will present some important recovery methods that give some solution to repair a happened failure to maintain the continuity of the execution.

2.3 Recovery of failures

To overcome the failures that happened when composing web services, various methods have been proposed to recover from these failures. The work in [1] has classified mainly recovery methods into two sections; backward error recovery and forward error recovery. Backward error recovery involves rolling back to a safe state, and retrying the operation. While forward error recovery consists in trying to recover from an erroneous state by transforming it into a safe state.

Among the important recovery methods, we present:

- Transactional Approach: [3] defines a transaction which is an operation that has an all-or-nothing ACID property which means Atomic, Consistent, Isolated and Durable. Transactional Approach can be used to recover from failures that can occur. The basic idea behind a transactional approach is: Only commit when every sub goal has completed successfully.
- Dynamic Web services composition approach occurs during runtime. Services are bound to other services on the fly (based on their WSDL descriptions and ontological annotations). Dependency and composition failures can easily be solved by this method [1].We can make use of other recovery methods, in conjunction with dynamic composition, to solve these problems more effectively.
- The study in [4] uses a system dynamics approach to model an e-service recovery framework for e-trust. The results of this study based on the complex recovery process. This study uses conduct simulation to evaluate the recovery performance based on the system dynamics approach.

Having now defined some types of failures and having also presented some of the most recovery methods. We need now to detect whether a failure occurred or not. For that, there is several failure detection algorithms are used to detect if something wrong happened during the composition of the web services. We distinguish two main categories [12] dynamic detection of error, which the error is detected during the execution of the web services or at runtime and Static detection category that the errors or failures aren't detected during run-time.

3. RELATED WORK

In [5], Bishop et al. have proposed fault taxonomy related to the Web service composition with a brief explanation of causes and consequences of faults. The recovery mechanisms are far from complete. They proposed run-time monitoring and reactive strategies to ensure the correctness of the composition. However, they have only performed qualitative analysis on composition faults and quantitative analyses have not been considered. In addition, they don't present the failure and its impact on the composition recovery mechanism and also they don't handle the dynamic composition.

In [6], Moore et al. have presented an area of web service composition that has not received as much attention of dynamic error handling and re-planning, and enabling autonomic composition. The presented work focuses on crucial characteristics of autonomic composition which is a self-healing ability of the dynamically deployed composition system. They proposed also, a context-based fault handling strategies that efficiently determines remedies in terms of reuse of plans or AI-based replanning and subsequent plan conversion. This paper presents dynamic composition by taking in the account failure detection and recovery. Nevertheless, the implementation lacks performance due to conversion of Web service composition plan into an actual working service process. Plans generated are abstract instructions, whereas WS-BPEL is executable process language with binding and deployment information. This information are gathered, interpreted and converted to the correct format. This would include creating the WSDL files.

Poonguzhali et al. in [7] have proposed an approach to self-healing mechanism for dynamic web service composition. Self healing is a property of autonomic computing which makes the system to heal itself from the faults. They proposed architecture for self-healing which heals the QoS faults in dynamic web service composition. They gave procedure for the healing mechanism, which allows providing an alternate service for the faulty service by performance prediction. However, they are lack of detection failure, which in distributed service infrastructures is a necessity for reliable implementations.

In [8], Alodib et al. presents and implements a method which allows monitoring occurrence of failure in Service oriented

Architectures (SoA). The presented approach extends Discrete Event Systems techniques to produce a method of automated creation of Diagnoser Service which monitors interaction between the services to identify if a failure has happened and the type of the happened failure. However, this approach does not discuss how dynamic composition is done.

In [9], Yan et al. have presented a business process composed of distributed Web services. The interactions among selected Web services are based on message passing. To identify the Web services that are responsible for a failed business process is important for e-business applications. Therefore they developed a monitoring and diagnosis mechanism based on solid theories in Model-based Diagnosis. Automata are used to give a formal modeling of Web service processes described in BPEL. But, the approach is more complex to extend a BPEL engine since it based on both Artificial Intelligence (AI) and Control Theory communities and also it lacks of failure recovery.

Yu et al. in [10] have presented a broker-based framework for dynamic and adaptive QoS-Aware Web services composition with QoS constraints. They have designed service selection algorithms for both the optimal selection and multiple adaptive secondary selections to be used when a service cannot meet its SLA at runtime. However, the selection algorithms can only handle one QoS constraint and a single point of failure, but they don't present either the failure detection or recovery.

Moser et al. in [11] propose an interesting adaptation technique coming from workflow based on execution monitoring. Especially, this approach allows monitoring of BPEL process according to QoS and replacement of existing partner service, by intercepting SOAP messages to enable services to be exchanged during runtime. They presented VieDAME (Vienna Dynamic Composition and Monitoring Extension) framework that uses a selector component to choose the most adequate service. In this framework, each service in a BPEL process can be marked as replaceable to indicate an alternative service that can be configured and invoked instead of the original service. The weakness of this approach lies in the lack of detection failure.

Christos et al. have presented in [12] an approach for dynamically resolving exceptions occurring in WS-BPEL scenario executions. This approach caters of the resolution of exceptions generated due to system faults (e.g. network fault) by proposing the Alternative Service Operation Binding (ASOB) Framework. This framework represents a middleware based approach for dynamically intercepting any failures and resolves them by invoking operational replacement services that are equivalent to the failed one. Nevertheless, this approach doesn't support recovery failure.

4. OUR CONTRIBUTION

In this section, we describe how our flexible approach can be used to enhance the mechanism of detecting and revering a failure at runtime.

When a failure has been detected by the system during the execution of the services composition, a recovery algorithm is executed dynamically which enables to replace the failed service by the suitable one by intercepting SOAP messages and making transformations to incoming and outgoing SOAP messages to adapt them to the new service that will be used in the BPEL process.

4.1. Description of our architecture

We present a recall of our architecture that was already described in our previous work [13]; this architecture (cf. Figure1) is modularized into five mains components:

- Request Analyzer: enables to analyze the client's query to identify his needs in order to specify the required functionalities in term of an elementary Web services.

- Discovered component: allows to discovered and locate the Web services responding to the user request and published in the Web services registries.

- Constraints Manager: is responsible to monitor the availability of the critical resources (CPU, network, service available,..) at runtime, and to detect if a failure occurs during the workflow execution. Also, to detect a change in user preferences, after that it will notify the Selection Manager component.

- Selection Manager: plays an important role to generate or regenerate the execution plan, from a communicated list of available services discovered in Web services registry and in harmony with information received from Manager Constraints.

- Orchestrator: has the ability to handle the execution plan in BPEL language of the generated Web service composite.

Our contribution is limited to the three components in the red box which are: the Constraints Manager, the Selection Manager and the Orchestrator.

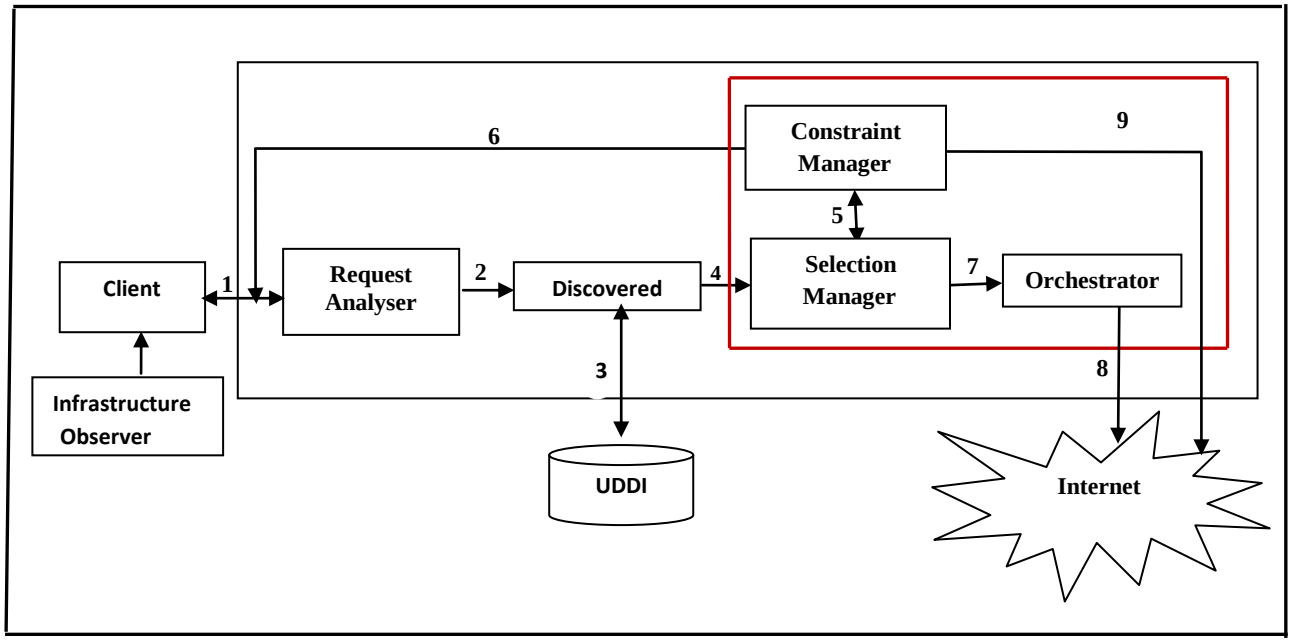

Fig 1: The proposed architecture

4.2. Framework for monitoring and recovering failure at runtime

To realise the adaptation for the failed web service in the workflow system, we propose a framework which has its architecture as shown in (cf. Figure 2). The framework contains the following main components:

- The Constraint Manager Module directly communicates with the user profile through the network. There are two components associated with the constraint manager; the BPEL Verifier and the BPEL Adapter:

  - ***The BPEL Verifier*** has the capabilities to dynamically detect an error which implies that the failure is detected during execution or during run-time. As an example of failure handled times and considering the time-out detector of failure that can be detected in such a way by encapsulating the invoke action in a scope that has a timer. Once the timer has run out, the service can recover from the time-out exception by calling another service, or retrying to invoke the same service. As we know that the BPEL orchestrating Framework couldn't control in advance if a partner service becomes unavailable or faulty, a possible workaround to solve this lack is using monitoring capabilities to predict and anticipate this failure using this component to examine and validate the partner service invocation by analyzing SOAP calls that results from <invoke> activities in the BPEL process. The idea is based on intercepting SOAP requests sent from the Orchestrator before being received by a corresponding partner service.

  - ***The BPEL Adapter*** is called when the partner service invoked is unavailable, it aims to locate and substitute at runtime the failed partner links by injecting available alternative one and thereafter to build the new SOAP request and sent it for execution. BPEL adapter applies XSLT 2.0 [14] transformations related to execution context and user's requirements.

- User's requirements and Execution Context allows to collect current context information.

- Selection Manager Module which is stored discovered web services that response user profile.

- Workflow Description is the document that describes the workflow processes BPEL format.

- Orchestrator Module is capable of parsing the workflow description by checking the availability of the partner service involved in the business process by interacting with the corresponding service in the Selection Manager Module.

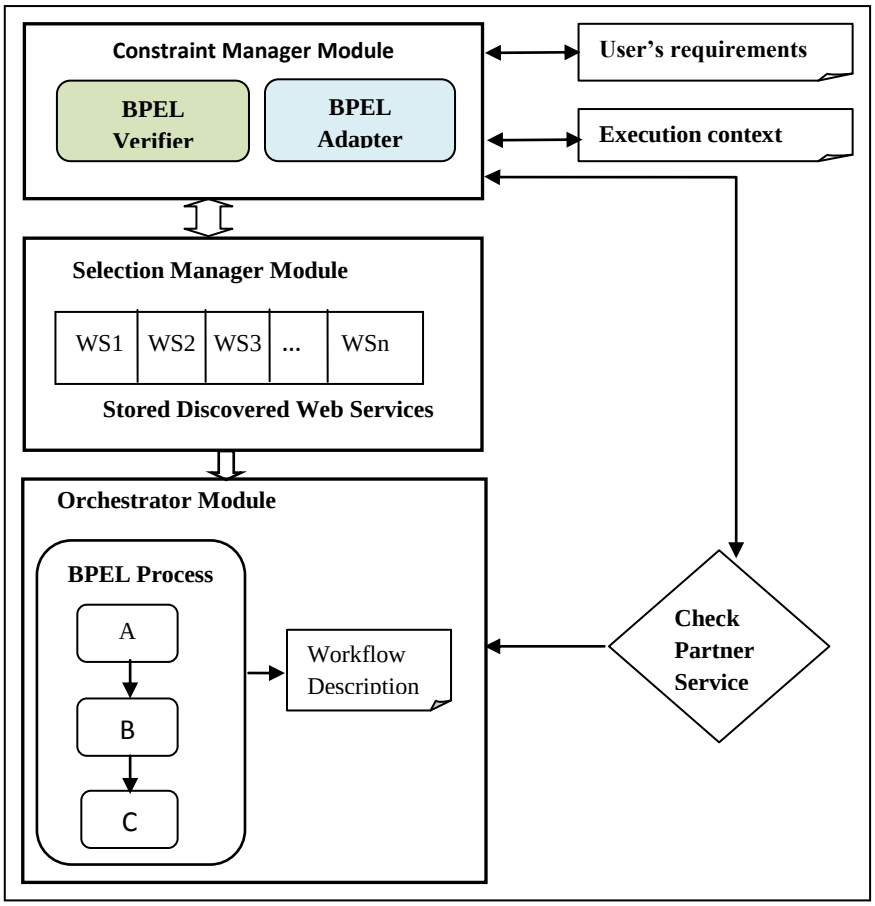

Fig 2: The architecture

4.3. Behavior of the architecture

In our proposed system, the Constraints Manager is a service which is invoked at each activity node of the BPEL workflow, by checking the availability of the partner service (WSi+1) according to the user's context by performing simple test verification by the BPEL Verifier to ensure the corresponding service available if a failure occurs the BPEL Adapter will perform the failure recovery. The Figure 3 represents the process of failure detection and the recovery process.

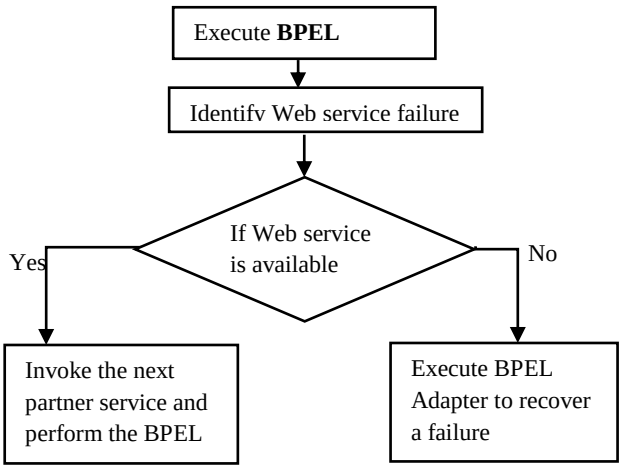

Fig 3: The runtime failure detection

The BPEL Adapter ensures to recover the failure once it has been detected. Firstly, we test if there is any change in the user's context, if the web service (WSi+1) is failed, we check the list that contains the best Web service selected that fulfil to the user's requirements, if there is one alternative web service that has the same functionality as the failed one in the Selection Manager module, and then substitute the web service for the failed service. If we found several alternative web services, the user has the ability to make a choice based on some web service criteria's as CPU, memory, network bandwidth..), in the case we don't find any service replacement an answer will be displayed to the user informing him that the specified service is not available and there is no alternative service. Figure 4 represents the recovery process.

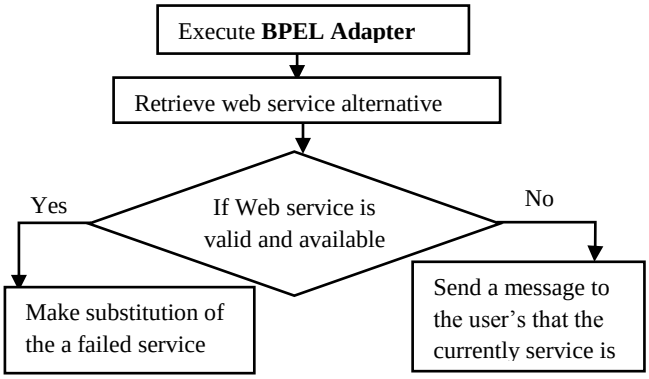

Fig 4: The recovered failure processes.

We proposed the following algorithm (cf. Figure 5) in pseudo code to retrieve the valid web service alternative, described in our previous work [13].

```
1   /* Variables list*/
2   wsInvoked /*originally invoked web service*/
3   userconstrainteslist /* user profile contraints
4   finRS /*final result*/
5   wsInvoke← getPart(request, "web service");
6   if  isAvailable(wsInvoke){
7   for (int i = 0; i < userconstrainteslist.getLength();
    i++) {
8   if (isvalid(wsInvoked ,userconstrainteslist(i))
9   {
// BPEL verifier evaluate if the originally invoked service
//valid each profile constraints
10  finRS← true; // the service is available to be
    executed
11  }
12  else {
13  finRS←false; //the service is unavailable the
    //execution will failed and thus the BPEL adapter is
    //invoked to substitute an alternative service for the
    //failed service
14  }
15  }
16  }
```

Fig 5: BPEL verifier and adapter Web service invocation

4.4. Prototype Implementation of our architecture

We have implemented a prototype of our architecture that includes the BPEL workflow implemented using java language (jdk1.6.0_07) on a Windows XP. Tools we have used are: Eclipse BPEL Designer to model, to design, and to build composite services, the Orchestration Director Engine (ODE) server for deploying and monitoring business processes, Tomcat 5.5 and Axis 2-1.2.

The BPEL Adapter replaces the failed service by the new alternative service base on the soap message as seen below:

We have used the "SoapExternalService.java" class of ODE and overriding the invoke() method by calling the "BpelVerifierService.java" and using this line we can monitor soap message:

SOAPEnvelope soapEnv = mctx.getEnvelope();

We read the soapenv and we change the soap message using the method called updateSoapEnv().

```
public void invoke(final PartnerRoleMessageExchange
odeMex) {
    boolean isTwoWay =
odeMex.getMessageExchangePattern() ==
org.apache.ode.bpel.iapi.MessageExchange.MessageExchang
ePattern.REQUEST_RESPONSE;
    try {
      ServiceClient client = getServiceClient();
      // Override options are passed to the axis
MessageContext so we can
      // retrieve them in our session out changeHandler.
      final MessageContext mctx = new MessageContext();
      /* make the given options the parent so it becomes the
defaults of the MessageContext. That allows the user to
override
      * specific options on a given message context and not
affect the overall options.  */
      mctx.getOptions().setParent(client.getOptions());
      writeHeader(mctx, odeMex);
      _converter.createSoapRequest(mctx,
odeMex.getRequest(), odeMex.getOperation());
      SOAPEnvelope soapEnv = mctx.getEnvelope();
      String mexEndpointUrl =
BPELVerifierService.verifierURL (
                                    (MutableEndpoint)
odeMex.getEndpointReference()).getUrl());
      EndpointReference axisEPR = new
EndpointReference(mexEndpointUrl);
```

A summarized code of BPELVerifierService is presented as follow:

```
public class BPELVerifierService {
 static String verifierURL(String url){
if(!isavailable(url)){
// lookup for the alternatif url
return  lookupAlternatifURL(url);
} else return url; }
private static String lookupAlternatifURL(String url) {
      String urlfind;
      return urlfind;
      }
private static boolean isavailable(String url)
 // check if the service is available
return false;}}
```

We verify the partner service invoked if its available the service is invoked and executed. Otherwise, we replace it by the available alternative service by calling the BPEL Adapter and execute it.

5. COMPARISON BETWEEN EVOKED APPROACHES

The table (cf. Table 1) below compares the different evoked approaches for Dynamic Web service composition in related work section and our framework by summarizing their characteristic. This comparison is made by using the most important criteria consisting in execution monitoring, QoS modeling, recursive composition, and composition/execution failure recovery and support user interaction.

6. CONCLUSION

Our work aims to investigate an approach to monitor and recover dynamically composition plan of web services. This approach has the advantage to compose web service on the fly and detect service failure, locate it and replace it at runtime by an alternative service implementing the same functionality according to user's profile.

We are currently working to extend our solution to support other failures type related to the context of execution of BPEL plan. In addition, a second experiment is in progress, which is related to a test the overall system in order to review the process of detection and recovering according to different user's contexts.

Legend: x = no support, √ = full support

Table 1.Comparison of requirements supported by different composition frameworks.

	Service Framework								
#	**Requirem-ents**	**Auton-omic service compos-ition**	**Self healing approach**	**COMPS**	**TGSE**	**Broken-Based Frame-work**	**VieD-AME**	**ASOB**	**Our**
-	**Language used**	OWL-S / BPEL4WS	BPEL4WS	BPEL4WS	BPEL4WS	BPEL4-WS	BPEL4-WS	BPEL4-WS	BPEL4-WS
1	**Dynamic composition**	√	√	x	x	√	√	√	√
2	**On-the-fly recomposition**	x	√	x	x	√	√	√	√
3	**User interaction**	x	x	x	x	x	x	x	√
4	**Automatic service discovery**	x	x	x	x	√	x	x	√
5	**Monitoring**	V	√	√	√	√	√	√	√
6	**QoS Modeling**	x	√	√	√	√	√	√	√
7	**Composition failure recovery**	√	√	√	x	√	√	√	√
8	**Execution failure recovery**	reactive	x	reactive	reactive	reactive	reactive	reactive	proactive and reactive

7. REFERENCES

[1] Mohammad Reza Motallebi,"Failure Recovery Web Service Composition", June 2010.

[2] Petrus Johannes Steyn,"Approaches to Failure and Recovery in Service Composition", November 2006.

[3] M.Xiaoyong,L.Shixian and H.Changqin, Transactional Dependency for Failure Recovery in Web Services Composition System, Chinese Journal of Electronics. Vol.21, No.2, Apr. 2012.

[4] Wei-Lun Chang, Hui-Chi Chang,, A Dynamic System of E-Service Failure, Recovery, and Trust, AIS Electronic Library (AISeL), 2010.

[5] J.Bishop,K.S. Chan,J.Stey,L.Baresi,S.Guinea,(2009)" A Fault Taxonomy for Web Service Composition ",Springer-Verlag Berlin, Heidelberg .

[6] C.Moore,M.X.Wang and C.Pahl,(2010)"An Architecture for Autonomic Web Service Process Planning",Emerging Web Services Technology Volume III, Whitestein Series in Software Agent Technologies and Autonomic Computing, Volume . ISBN 978-3-0346-0103-0. Birkhäuser Basel, 2010, p. 117

[7] S.Poonguzhali, R.Sunitha, Dr.G.Aghila, (2011) "Self-Healing in Dynamic Web Service Composition",International Journal on Computer Science and Engineering (IJCSE).

[8] M.Alodib and B.Bordbar,(2009) "A model-based approach to Fault diagnosis in Service oriented Architectures",Web Services, 2009. ECOWS '09. Seventh IEEE European Conference.

[9] Y.Yan, P.Dague, Y.Pencol and M.O.Cordier," Testing Service Composition Using TGSE tool ", International Journal of Web Services Research JWSR (2009).

[10] T.Yu and K.J.Lin,(2009),"A Broker-Based Framework for QoS-Aware Web Service Composition".

[11] O.Moser, F.Rosenberg and S.Dustdar," Non-Intrusive Monitoring and Service Adaptation for WS-BPEL", International World Wide Web Conference Committee (IW3C2). WWW 2008, April 21–25, 2008, Beijing, China. ACM 978-1-60558-085-2/08/04.

[12] K.Christos, C.Vassilakis,E.Rouvas and P.Georgiadis, "Exception Resolution for BPEL Processes: a Middleware based Framework and Performance Evaluation", iiWAS2008, November 24–28, 2008, Linz, Austria. 2008 ACM 978-1-60558-349-5/08/0011.

[13] K.Boumhamdi,Z.Jarir,"A Flexible Approach to Compose Web Services in Dynamic Environment", International Journal of Digital Society (IJDS), Volume 1, Issue 2, June 2010

[14] W3C. XSL Transformations (XSLT) version 2.0, http://www.w3.org/TR/xslt20/ (Last accessed 2011).