Policy-based Exception Handling for Service Processes

Xue Tong, Ying shi, Wenjing
State Key Laboratory of Software Engineering
Wuhan University
China 430072

Wangquan-yu
Distance Education College
China University of Geosciences
China 4000742
yingshi@whu.edu.cn

ABSTRACT: *Due to the lack of an efficient separation mechanism between the business logic of the service process and the business logic of exception handling, there are still many worthy of study in exception handling mechanism of BPEL process. For instance, the flexibility of the exception handling, the maintainability and the readability of service process. Aiming at improving the exception-handling capacity of BPEL process, this paper explores a policy-based exception handling method for BPEL processes, and proposes a policy-based exception handling description language WS-Policy4BPEH, and designs a framework to support the implementation mechanism of exception handling process based on WS-Policy4BPEH. Finally, through an experiment, this paper demonstrates the policy-based exception handling approach for BPEL processes .*

Keywords: Policy, Service Processes, Exception Handling

Received: 22 October 2016, Revised 29 November 2016, Accepted 2 December 2016

1. Introduction

In recent years, how to improve the dependability of the service process stands out as one of the key scientific problems in service-oriented software engineering [1]. In the aspect of the exception handling of the service process, an internal exception handling mechanism has been successfully implemented in the service-oriented software development language, but there have not been enough researches and there still exists many issues worthy of research, for instance, the flexibility of the exception handling, the maintainability and the readability of service process. The fundamental reason for these issues lies in the lack of an efficient separation mechanism between the business logic of the service process and the business logic of exception handling.

Policy refers to the principles or rules that are used to guide the decision and adjust system. Policy–based management is a management model that separates the system management actions and system application functions [2]. It has been widely researched and used in network management, security management and so on. A policy–based exception handling method for service process is proposed in this thesis, which utilizes policy to describe the logic of exception handling of service process, realizing the separation of the focus points of the exception handling logic and business logic of service process. As a result, the flexibility of the exception handling of the service process has been tremendously enhanced.

2. Relevant Researches

Zeng et al propose a policy-driven web service composition exception management framework, using policy to refer to the knowledge on exception handling of service process. The exception handling policy is based on ECA principle paradigm, which indicates that in the case of an exception in the service process (Domain, On), when the conditional restraints are fulfilled (When), the exception heal action should be taken (Do) [3]. The methods provided by Liu An et al are similar to above, but the difference lies in that the former has given a description method on the exception handling principle of the service process based on the ECA paradigm and defined multiple exception handling patterns, such as Ignore, Retry and Skip, etc [4]. Aiming at the limitation of the credibility support that the current BPEL can deliver for the service process, A.Erradi et al put forward a method for the self-adaptability of the policy-based service process [5]. L.Baresi et al proposed a Self-Supervising BPEL Processes to boost the capacity of service process exception handling [6]. K.Kim et al raised a rule-based service process proactive exception handling approach, which is used to deal with exception according to the exception handling lifecycle, based on the action, function and information requirements of the service process [7].

Based on the Self-healing theory, G.Friedrich et al came up with a service process Self-healing method: defining the set of action of the service process exception handing, providing the process reparability analytical procedures during the design stage, and producing a generation method for exception handling during the operation of service process based on the AI planning theory and according to the service process structural constraints, data dependencies among the service process activities, and the available exception handling action analysis [8]. L.Baresi et al proposed a service process self-healing method-Dynamo, which is a middleware based on assertion, and adopts WSCol language and WSRel language to define the monitoring and recoveryabilities of the service process. Dynamo is an extended ActiveBPEL engine that uses JBoss rule to realize the self-healing ability of the service process [9].

S.Modafferi presented a service process exception handling method based on annotation, which required the service process developer to add annotation on exception handling action during the design stage and produce a standard fault-tolerant service process through the pre-handling of the annotation service process in the deployment stage based on BPEL [10]. To solve the partner service exception of the service process, O.Ezenwoye et al brought forward a service process exception handling method based on proxy, which used equivalent service to replace breakdown service, realizing the service process exception incurred by the partner service breakdown [11,12].

3. Service Process Exception Handling

Service process exception refers to the deviation of the service process operation from the service process design. Exception handling is the process of elimination the “deviation” by bringing the service process from the Service process exception refers to the deviation of the service process operation from the service process design. Exception handling is the process of elimination the “deviation” by bringing the service process from the abnormal track back to the normal track.

3.1 Classifications of Service Process Exceptions

To classify the service process exceptions from the perspective of fault sources is beneficial to analyzing the causes of exception and taking targeted handling measures. This paper systematically lists the various types of exceptions that may appear in the service process execution from the three aspects of system exception, resource exception and application exception.

System Exception refers to the exception caused by the breakdown in soft and hardware infrastructure when the service process is running, and is often sub-divided into hardware exception, software exception, and communication exception. ① Hardware Exception refers to the exceptions triggered by the failure in such equipment as the server and the storage, which can be prevented through hardware redundancy. Once a hardware exception happens, the service process operation must be suspended or terminated for manual handling. ② Software Exception refers to the exception caused by the malfunction of such

software as the OS, database, middleware, which can be prevented through software redundancy, but also requires manual handling. ③ Communication Exception refers to the failure to access resources attributed to network fault, which can be prevented through link redundancy. For temporary communication exception, a retry method may be applied.

Resource Exception refers to the exception brought about by the fault in the data resource and computing resources required when the service process is running. For the service process, computing resource is the partner service, whose failure is the major source of resource exception; therefore, the resource exception of the service process mainly refers to the service resource exception. Resource exception is usually reflected in the unavailability of service resource, exception of service resource content, mismatch of service interfaces, service protocol binding errors, service QoS exception, SLA conflict. ① Unavailability of service resource refers to the failure to access the WSDL documents of Web service, for instance, the service provider updated service UPI, which makes the service user unable to gain the description information on the service. ② Mismatch of service semantic happens when the service resource doesn't give a correct response, which is caused due to internal design logic error, thus belonging to design exception. ③ Mismatch of service interfaces refers to the mismatch of the actual interface of web service and the WSDL-defined interface, such as the input parameter types mismatch and the input parameter numbers mismatch. This type of exception also needs the service provider to update the service WSDL documents, therefore also belong to design exception. ④ Service Qos exception happens when the web service can normally fulfil the described service function but does not match with the QoS defined by WSDL. ⑤ SLA conflict refers to the discrepancy between the non-functional properties and the quality protocol of web service, such as the higher service fees paid than prescribed in the service agreement.

Application Exception refers to the exception incurred by the application logic error of partner service or service process, and is often sub-divided into service application exception and process application exception. ① Service application exception refers to the exception that is incited by the dissatisfaction in the service protocol, such as the full hotel reservation exception by the hotel reservation service. ② Process application exception refers to the exception caused by the failure of contextual information to meet the process application logic or constraint. Application exception belongs to user-defined exception and is defined by the web service and service process designer during the design stage. Nevertheless, it is impossible to predict all the application exceptions in the design stage, therefore, "Anonymous Exception" is defined to indicate the unpredictable exceptions.

3.2 Exceptions Handling Actions of Service Process

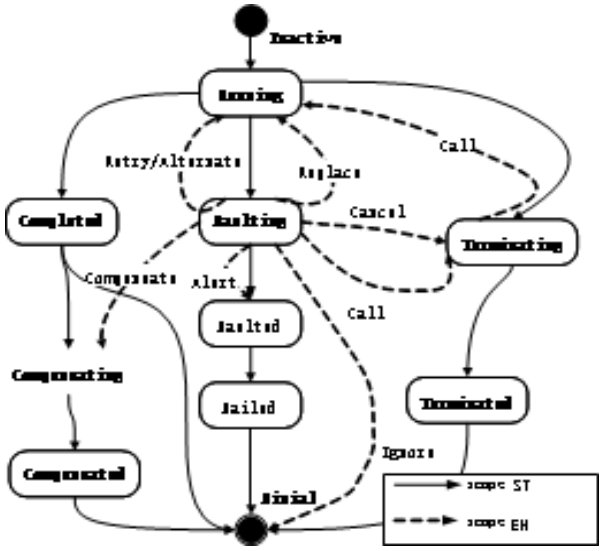
Exceptions handling action is the abstraction of the exception handling actions. Exception handling actions are divided into Atomic Action and Composite Action. Atomic actions cannot be subdivided, indicates basic exception handling behaviors; Composite Action forms by Atomic Action or Combination Action according to support for complex exception handling logic.

3.2.1 Atomic Action

Atomic Action, the abstraction of the usual service process exception handling action, is an invisible exception handling action. In WS-BPEL standard, scope is the container of exception handler. The invoke activities to call partner service and the service process are essentially a hidden scope, whose lifecycle is composed of different statuses—from Inactive status at the beginning to Running after being activated by service process engine to execute scope activities . During the running process, scope may also enter into various statuses: ① Completed, which means that the service process engine can continue the follow-up activities after scope successfully finishes the process logic; ② aulting, which means that exceptions are trigger due to the fault in running environment, process design or partner service resource, and scope activities begin exception handling; ③Terminating, which means that scope activities are terminated. ④ compensating, for the parent scope that has sub-scopes, it can compensate the completed sub-scope activities to guarantee the consistency of the statuses of service process. The statuses of scope activities are shown in figure 1.

When scope is in faulting, it throws a fault message that triggers an exception, which then leads to the exception handling action. There may be multiple exception handling choices for scope exception, with each corresponding to a different exception handling action. Based on the existing researches, this paper, in accordance with the service process exception handling requirements, divides the exception handling atomic action into the 9 categories of Ignore, Skip, Retry, Alternate, Replace, Cancel, Compensate, Call and Alert.

Ignore means that the service process ignores the exception and continues to execute the subsequent activities beyond the failure domain. All the activities in the failure domain are to be terminated compulsorily and the service process jumps from the faulting status to completed status.



Figire 1. Scope Activity Status

Skip means that the service process skips the subsequent activities As in failure domain and continues to execute the process. From the perspective of control flow, the service process skips As to its subsequent activity As to continue the execution.

Retry means when the failure domain throws out an exception message, the service process repeatedly executes the failure domain until the activities are successfully completed or reach the defined retry times or deadline.

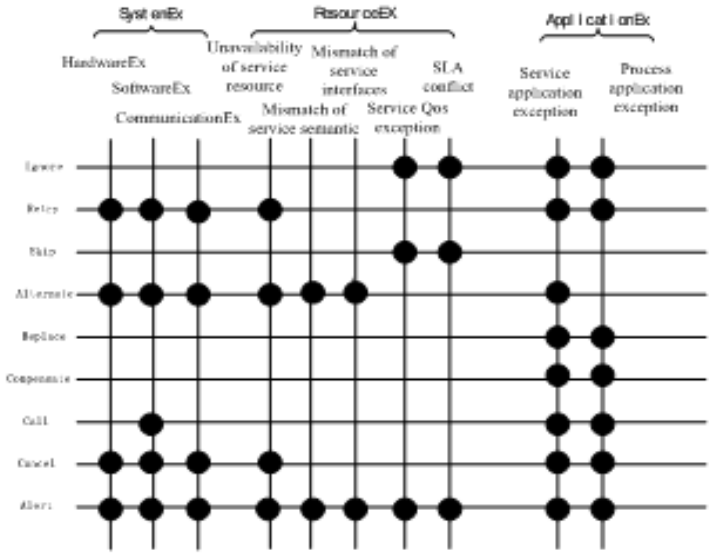


Figure 2. Service Process Exception-Action Relationship

Alternate means to bind the equivalent services {WS1,WS2,…,WS}from the partner community in the sequence order until success or the completion of all equivalent services.

Replace means that if Activity1 and Activity2 are equivalent activities, then Activity2 can be used to replace Activity1 when an exception happens in the failure domain Activity1.

Compensate means when an exception occurs in the failure domain, and the inserted scope1, scope2…, scopeN have been completed, then compensation may be made in the reserved order, from scopeN, scope n-1,… scope1.

Cancel means to call off the execution of faulting activities immediately

Call means to invoke external service (like suspension, recovery and termination, etc.) to handle the service process instance exception.

Alert means to inform the service process management of the exception or to record the exception information in the log.

The service process exception handling actions are subject to the exception types detected. A service process exception-action matrix can be derived (as shown in Figure 2) through the analysis of the service process exception model and exception handling action to guide the development of the exception handling policy.

3.2.2 Composite Action

Composite Action is formulated by the integration of atomic actions or combined actions according to different control structures, used to indicate complicated exception handling logic. At present, the composite action supported by the WS-Policy4BPEH (next chapter) include the EHSequence, EHFlow and EHSwitch, all of which will be illustrated from the aspects of action purpose, usage scenario, action definition and execution semantics as follows.

EHSequence

① Action Purpose: to execute exception handling action, whether atomic action or composite action according to the sequence combination. EHSequence is the embodiment of an action redundancy fault-tolerant mechanism.

② Usage Scenario: if activities a1 and a2 are compatible, then the sequence order of a1 preceding a2 or the other way around can be executed.

③Action Definition:

EHSequence: $= action_1 > action_2 > ... > action_n$, action1, action2, ..., actionN are atomic actions or composite actions.

④ Execution Semantics: to use the action sequence indicates the exception handling action se— EHSequence = {action1, action2, ..., actionN}. The actions are executed in order, and if the exception is solved after the first action is executed, then exception handling is finished, the subsequent actions are skipped and the sequence action activity is completed; or else, the following actions need to be executed until the exception is successfully handled or all the actions are all executed.

EHFlow

① Action Purpose: to concurrently execute the exception handling actions; EHFlow is also the embodiment of an action redundancy fault-tolerant mechanism, but has better performance than EHSequence, which may come at the cost of credibility.

② Usage Scenario: if actions a1 and a2 fulfill the conditions in Assertion1, then actions a1 and a2 are concurrently executed.

③ Action Definition:

EHFlow: $= action_1 \square action_2 \square ... \square action_n$

④ Execution Semantics: to concurrently execute exception handling actions action1, action2, actionN, and if one of them succeeds, then the composite action is finished and all the other actions are terminated.

EHSwitch

① Action Purpose: to choose an exception handling patch according to the contextual information of the exception when the service process is running; this action is used to improve the adaptability of the service process exception handling.

② Usage Scenario: the choice of exception handling patch is dependent on the contextual information of the exception when the service process is running, therefore, the contextual information needs to be gained before selecting branch conditions and match it with qualified exception handling patch.

③Action Definition:

EHSwitch: = (*condition*, $action_1$, $action_2$) condition refers to the selection conditions and action1 refers to the exception handling action when the conditions are real, while action2 indicates the exception handling action when the conditions are false.

④ Execution Semantics: to select an exception handling action according to conditions.

4. Service Process Exception Handling Policy Description Language WS-Policy4BPEH

4.1 WS-Policy4BPEH Concept Model

Through the research and analysis on the service process exception handling mechanism, this section, taking reference from existing policy description language, provides a WS-Policy4BPEH language Concept Model (as shown in FIGURE 3). The top-level design of this model includes such core elements as EHPolicySet, EHPolicy, EHRule and EHAction.

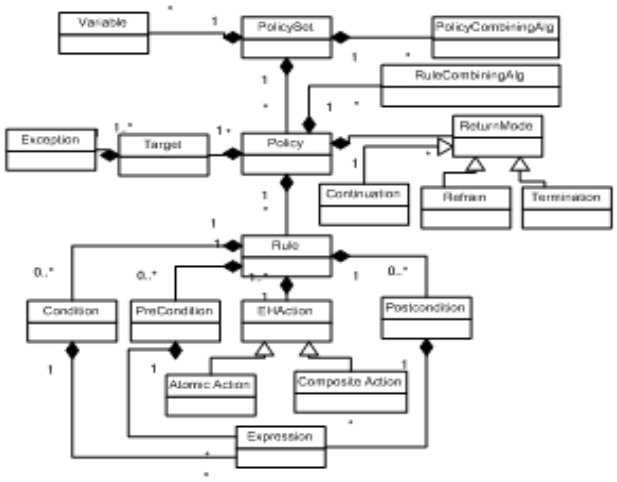


Figure 3. WS-Policy4BPEH Language Concept Model

EH PolicySet is the set of service process exception handling policies that are independent and unordered. It describes at a higher level of abstraction how to guide and control the service process exception handling actions. Within the set, policy variables can be defined to indicate the process status and system environment during the service process exception handling. Policy Combining Alg defines the resolution methods in the case of a conflict among different policies within the set.

EH Policy is the set of rules. In the WS-Policy4BPEH language, policy, lying between policy set and rules, is not only the basic integral unit of policy set but also a container of exception handling rules. Policy target element defines the types of exceptions that can be handled by policy, which is the sequential combination of rules, i.e. if the previous rules fail, the subsequent rules are executed until success or the completion of all rules. If the policy execution fails, then the service process exception handling action should be controlled according to the policy return method.

EH Rule expands the ECA rule paradigm, while the WS-Policy4BPEH rule defines the response to service process exception. Its execution semantics is stated as follows: when there is an exception in service process, if the service process status fulfills the conditions of the rule as well as the preconditions for the action, then the exception handling conduct should be executed, after which the post-conditions of the rule shall be calculated to see whether it meet the bundle of the successful exception handling.

EHAction defines the exception handling actions of the rule, which includes atomic action and composite action. At present, the atomic action defined include Ignore, Retry, Replace, Skip, Alternate, Compensate, Cancel, Call and Alert, while the composite action comprises atomic action or other combined actions such as EHSequence, EHFlow and EHSwitch, the description of which can be found in section 3.2.

4.2 The Syntactic Structure of WS-Policy4BPEH

WS-Policy4BPEH uses XML as its meta language for its rigorous syntactic format, availability, expandability, transportability, and it can supported by many tools. As a service–oriented process exception handling policy description language that not only has the basic elements and properties of universal policy description language but also fully demonstrates the characteristics of exception handling policy, WS-Policy4BPEH language supports the description of different levels of abstraction from exception handling action to rule, policy and policy set. One service process model is equipped with one WS-Policy4BPEH policy document that contains at least one service process EHPolicy Set which is a non-blank set of EHPolicies. Policy is the non-blank set of EHRule, and the rules within the policy are executed according to the sequential combination. Exception handling rule expands the ECA rule paradigm, and describes when there is an exception in service process, if the service process status fulfills the conditions of the rule as well as the preconditions for the action, then the EHAction should be executed and its effect evaluated. If all the rules within the policy fail, then the service process EHAction shall be controlled according the return method of the policy.

The general syntactic structure of WS-Policy4BPEH language is as follows:

```
<xs:schema targetNamespace=R2E>
<PolicySet PolicySetId=NCName PolicySetDefault=NCName?>
<Descriptions/>?
<Variables/>?
<PolicyCombingAlg/>
<Policy PolicyId=NCName Version=NCName? Priority=XS:Integer?>
    <Description/>?
    <RuleCombiningAlg/>
        <Target>
            <Exception>+
        </Target>
        <Rule RuleId=NCName DateTime=XS:DateTime? Priority=XS:Integer?>
            <Condition/>?
            <PreCondition/>?
            <EHAction>
            <PostCondition>?
        </Rule>+
         <ReturnMode/>
    </Policy>+
</PolicySet>+
</xs:schema>
```

5. Service Process Exception Handling Framework EHF-P Based on WS-Policy4BPEH

EHF-P Framework (as shown in figure 4) is an abstract, web service-oriented software framework based on WS-Policy4BPEH language, offering the general function for the service process exception handling. This framework not only sustains the detection, recognition, transmission and handling of the service process exception, but also provides a supporting environment for the design and realization of service process exception handling.

EHF-P Framework mainly includes the design environment, run environment and service resources set etc.

① EHF-P design environment: it provides the service process exception development with the model, analysis and deployment realization function of the service process exception handling logic based on EHP-P policy. The service process exception developer use the EHP-P Policy Editor to edit service process exception handling policy based on EHP-P, and use EHP-P policyto indicate the service process exception handling actions. To guarantee the correctness of EHP-P policy, the EHP-P policy Analyzer, based on the formalized semantics of EHP-P policy and the CPN Tools simulation analysis tool, can support the analysis and verification of the key properties of EHP-P policy.

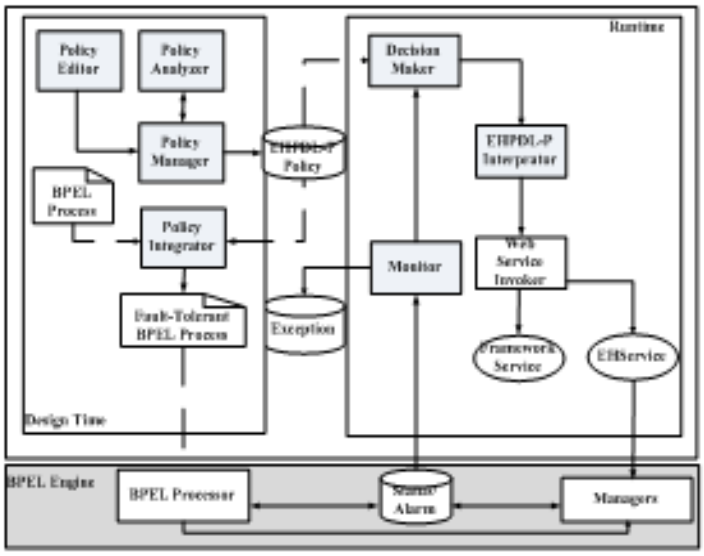


Figure 4. EHF-P Framework

② EHP-P Run Environment: it supports the dynamic configuration and realization of the exception handling logic of the service process. The Exception Detector can obtain the service process exception and running context from the service process engine and generate the service process exception handling requests that comply with the Policy Decision-Maker format. When the Policy Decision-Maker receives the exception handling requests from the Exception Detector, it will search the WS-Policy4BEH policy database and return to find a matching EHP-P policy for the exception. After receiving the EHP-P from the Policy Decision-Maker, the Policy Executor first analyzes the EHP-P to generate an EHP-P rule queue and then traverse the queue to generate exception handling action call through analyzing each rule.

③ Service Resources Set: a) System Exception Handling: the EHF-P framework provides such call service as suspension, recovery, termination for BPEL engine, middleware and operation system function to support the handling of some repairable system exceptions. b) Resource Exception Handling: Some resource exceptions may be caused by temporary or random reasons. The "Retry" method may be applied to handle such exceptions. For some exceptions caused by non-temporary errors, the following exception handling services may be applied: reserve service, retry after automatic interface is adapted or dynamically finding new available service. For some resource exceptions that requires manual handling, such framework built-in services as log and notification can be used to inform the framework users. c) Application Exception Handling: EHF-P framework provides the expansion mechanism, and supports the development of certain exception handling service (or service process language) for certain application exceptions. Such service is registered in the framework as a new component of exception resource set.

6. Experiment

	SP1	**SP2**	**SP3**	**SP4**	**SP5**	**SP6**
The number of services in services process	5	5	5	5	5	5
The number of exceptions were distributed	1	1	2	2	2	3
The number of exceptions were handled (without policies)	0	1	0	1	2	1
The number of exceptions were handled (using policies)	1	0	2	1	0	2
Handling time(without policies)(ms)		1.475		2.458	2.976	3.821
Handling time(using policies)(ms)	1.274		1.897	2.135		3.497

Table 1. Experiment Result

To test effects by using policy-based exception handling for service processes, we designed an experiment as follows. 6 groups of service processes were selected and each includes 5 services. Various amounts of exceptions were distributed proportionally into each service process in accordance with a one to one to one ration among hardware exception, software exception, and communication exception. Then, comparing the results of those with policies and without policies, we can clearly see the effects and data as table 1 shows.

7. Conclusion

This paper studied exception handling mechanism of the service process and proposed a policy–based service process exception handling method, which first put forth a service-oriented process exception handling policy description language WS-Policy4BPEH and then introduced, through analyzing the cases, the description method of the service process exception handling based on WS-Policy 4BEH. To support the development of the WS-Policy 4BEH service process exception handling logic, this paper also presented the exception handling framework EHF-P of the service process based on WS-Policy 4BPEH.

Our next step is to continue improving WS-Policy 4BPEH language and develop EHF-P framework tools to perfect the policy-based service process exception handling.

References

[1] Maoxinsheng. (2007). The principle, method and practice of SOA. Publishing House of Electronics Industry.

[2] Boutaba, R., Aib, I. (2007). Policy-Based Management: A Historical Perspectivc. *Journal of Network and Systems Management*, springer.

[3] Zeng, Liangzhao., Lei, Hui., Benatallah, B. (2005).Policy-Driven Exception-Management for Composite Web Services.Proceedings of the Seventh IEEE International Conference on E-Commerce Technology(CEC'05).

[4] An Liu, Qing Li, Liusheng Huang, Mingjun Xiao. (2007). A Declarative Approach to Enhancing the Reliability of BPEL Processes, *In*: IEEE International Conference on Web Services(ICWS 2007).

[5] Erradi, A., Maheshwari, P., Tosic, V. Recovery Policies for Enhancing Web Services Reliability, IEEE International Conference on Web Services(ICWS'06).

[6] Baresi, L., Guinea. S. A Dynamic and Reactive Approach to the Supervision of BPEL Processes, ISEC'08 39-48.

[7] Kim, K., Choi, I., Park. C. (2011). A Rule-based Approach to Proactive Exception Handling in Business Process. *Expert Systems with Applications*, 38. 394-409.

[8] Friedrich, G., Fuqini, M., Mussi, E., Pernici, B., Tagni, G. (2010). Exception Handling for Repair in Service- Based Processes. *IEEE Transactions on Software Engineering*, 36 (2) 198-215.

[9] Baresi, L., Guinea, S., Pasquale. L. (2007). Self-Healing BPEL Processes with Dynamo and the JBoss Rule, ESSPE 11-20.

[10] Modafferi, S., Conforti, E. (2006). Methods for Enabling Recovery Actions in WS-BPEL.OTM LNCS 4275, 219-236.

[11] Ezenwoye, O. (2008). Sadjadi, S.M. A Language-based Approach to Addressing Reliability in Composite Web Services. SEKE 649-654.

[12] Ezenwoye, O.S.M., Sadjadi.RobustBPEL2:Transparent Autonomization in Business Processes throught Dynamic Proxies.ISADS 2007:17-24