International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

TOWARDS FAULT HANDLING IN B2B COLLABORATION USING ORCHESTRATION BASED WEB SERVICES COMPOSITION

U. Arul [1], Dr. S. Prakash [2]

[1]*Research Scholar, Department of CSE, Anna University, Chennai, India.*
[2]*Professor, Department of ECE, Jerusalem College Engineering, Chennai, India.*

arulmee08@gmail.com, prakash.sav4@gmail.com

Abstract

Recently, Web Services Composition is receiving significant amount of interest as an important strategy to allow Business-to-Business collaboration. The current Web Services composition solutions are inherently unreliable, so how to deliver reliable Web Services composition over unreliable web services is a significant and challenging problem. In this paper, we propose a framework for reliable method of fault handling in B2B collaboration using Orchestration based Web services composition (WS-BPEL). The Fault handling method is to be applied during the web services composition process in order to perform some measure of exception handling. We developed a separate Fault handling module that could identify the faults and also that could handle the faults during the composition process. We also devised the various Exception handling strategies that could provides the series of action processing steps to execute alternative web service when a particular web service fails. Further more, we designed an implementation module to implement the reliable fault handling logic that could be associated with WS- BPEL business process.

***Keywords*— Web Services, Reliable Web services composition, Orchestration, Exception Handling.**

I. INTRODUCTION

Web services are playing an important role in the development of Business-to-Business collaboration. The Web services collaboration already implemented in order to provide new functionalities is an interesting approach for creating an enterprise applications, distributed applications and business processes. However, web services may run in a highly dynamic environment in an Internet that is existing web services may temporarily unavailable due to server fault, system crash and network failures etc. Such a highly unreliable environment increases the probability of deviation situations that occur during the execution of web services composition. So, it is important to provide a support for reliable service composition whenever the environment is changed. However reliable Business-to-Business interaction has not been investigated sufficiently and hence it is still an open issue [1].

To provide reliability in Web service composition is not considered as an easy task. WS-BPEL [2] which attempts to implement a commonly accepted way for defining web services composition, but it does not support reliability at runtime.

It defines a business process workflow that specifies the order of invocations (control flow) and rules for data transfer between the business partner web services (data flow). But the information about partner services, composition logic and data dependencies must be known at design time and it is clearly impossible for a composite service to avoid the faults during its execution [3]. Therefore, this paper aims at delivering a correct service in the presence of faults and it becomes a preferred choice for providing reliable web services composition.

Exception handling technique for reliable service composition tries to repair faults and let allow composition of services to continue [4]. The WS-BPEL has built-in exception handling mechanism. It provides scopes with fault handlers to handle faults, which is equivalent to handle faults in programming languages sch as JAVA using the try-throw-catch mechanism. However, service designers can develop reliable composite services by using exception handling we observed the following problems:

- The WS-BPEL constructs used to implement fault handling solutions that are located at syntax level. Hence, the business logic is associated with the fault-handling logic which makes it hard to implement and maintain both types of logic [5].

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

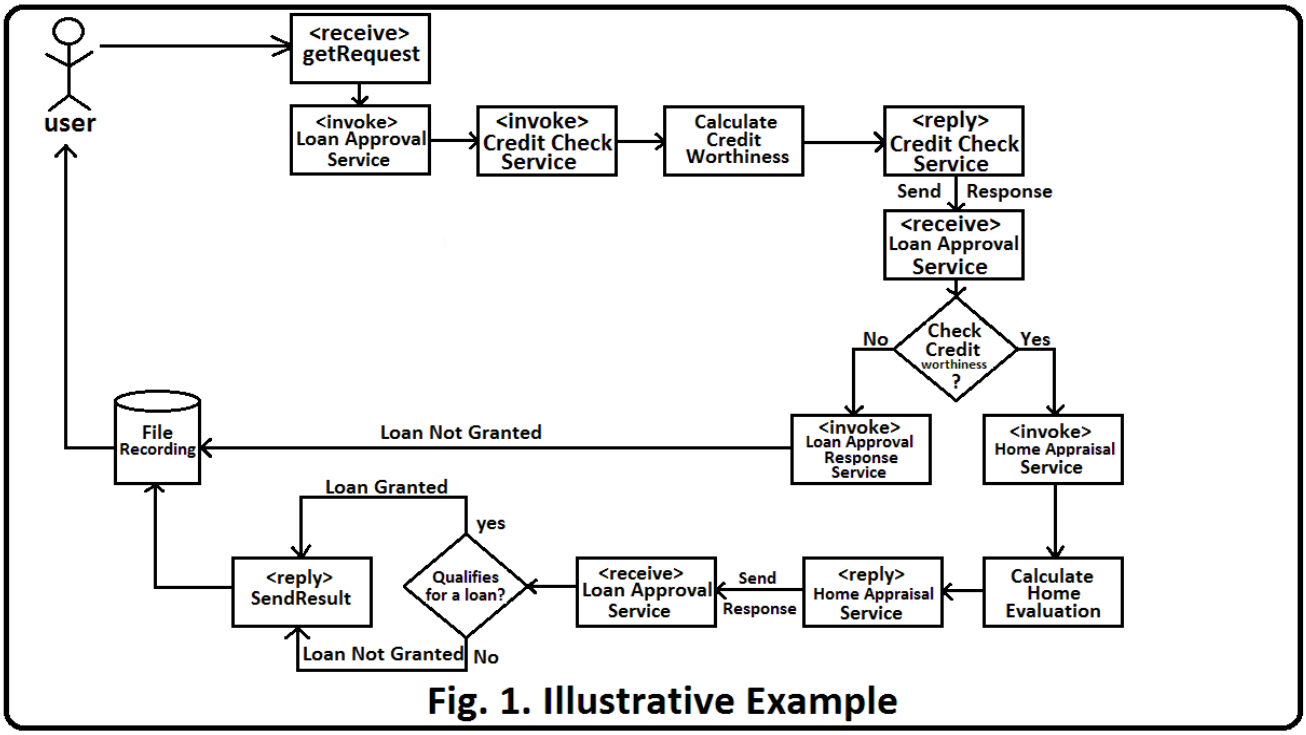

Fig. 1. Illustrative Example

- In practical situations, web services commonly cannot be compensated or the compensation is permitted only within a certain time period [6].

The rest of this paper is organised as follows: In the following section gives a brief discussion about the illustrative example of Loan Approval Composite service. Section 3 introduces Orchestration and WS-BPEL, this is followed in section 4 by a discussion of the proposed framework architecture. Section 5 gives an overview of faults in Web Services Composition. In Section 6 we discuss WS-BPEL and its existing fault handling methods. In section 7 we discuss current work on correctness of Web Service Composition that related to attaching fault handling logic to WS-BPEL. Section 8 presents some experiments to show the performance of framework model and we conclude the paper in section 9.

II. ILLUSTRATIVE EXAMPLE

Figure 1, refers to illustrative example, which shows a composite service that aims to verify whether to approve or reject the home loan request submitted by a customer. Our composite service gets a request from the customer. It then invokes Loan Approval web service and waits for response from it.

The Loan Approval web service in turn, which makes the invocation to Credit Check web service. The credit check service processes the received request message and sends back the information about credit worthiness of the customer to Loan Approval service. After receiving the response from credit check service the Loan approval service is verifying that whether the customer is credit worthiness or not. If the customer is credit worthiness, then the loan approval service invoke Home Appraisal web service, otherwise it sends the rejection of loan as a response message to composite service. The composite service records the message in its local database and sends it as a rejected loan application message to customer. The Home Appraisal service invoked for verifying whether the customer's house is worth the sum the loan, the customer is asking for. If then sends response back the result of the house evaluation to Loan Approval service. Upon receiving the response message from Home Appraisal service, which verify whether the customer is qualifying to get loan. After the verification processing is getting over, it sends the reply that is sending result message to composite service. The composite service stores the SendResult message in its local database also forwards the same message to the customer.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

III. ORCHESTRATION AND WS-BPEL

The process management of composite services can be managed in centralized or decentralized manner. In our paper, a centralized application that controls the process management of composite services and invokes external web services. This type of composition is usually called as Orchestration [7], and these external partner web services are as component services.

WS-BPEL [8] originated as a language from IBM, it is an OASIS standard to describe web services composition. There are two set of activities are compose a WS-BPEL business process, these are basic and structured activities. Basic activities exist to allow the invocation of an external service and to expose an interface to the process itself (to receive messages and send messages in reply), to assign values to variables and to signal faults.

Structured activities, which are basically control the execution of other activities nested inside it. Structured activities exist to allow sequential, parallel, conditional and looping execution (the sequence, flow, switch, pick, and while activities). WS-BPEL has relationship with WSDL. Interactions with partner services are modelled as PartnerLinks. A PartnerLink has a PartnerLinkType that defines which WSDL PortType is used in a relationship with some partner and which PortType is used when a partner interacts with the process itself. These two relationships are defined in the partnerRole and myRole attributes of the PartnerLinkType. For two way relationships both are used. An important aspect of using PortTypes means that WS-BPEL refers to services in an abstract way and an execution engine to determine what port (used for binding) should be used for each PortType. In common, the bindings can be specified statically at deployment time or dynamically by either from within the process or using some engine-specific mechanism.

IV. THE PROPOSED FRAMEWORK

A. *Layered Web Service Structure*

The framework represented here is based on a layered Web service architecture that is consisting of six different layers, which includes Transport layer, Message layer, Service Description layer, Service Assurance layer, Service Composition layer and Service Application layer, which is shown in Figure 2.

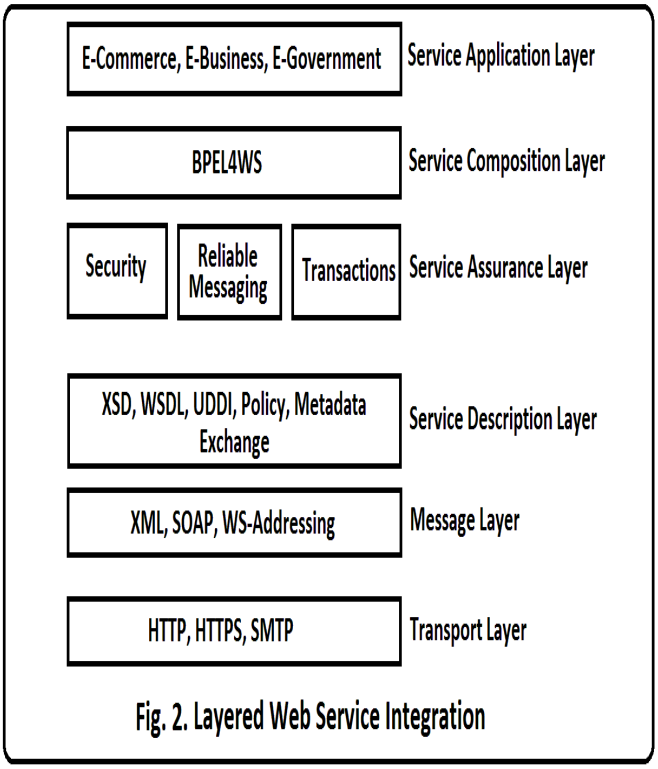

Fig. 2. Layered Web Service Integration

The bottom is the transport layer contains widely used communication protocols for communication, secure communication, E-mail messaging etc., over a computer network with especially wide deployment on the Internet. The message layer is on the top of transport layer which provides basic communication between web services and enterprise applications. The next top layer is the service description layer which provides the basic service description technology that is based on XML, such as WSDL, UDDI etc. The fourth layer is service assurance layer which facilitates WS-Security, Reliable Messaging among web services and WS-Transactions. The next top layer is service composition layer. On the basis of service failures and service unavailability, it is difficult to implement composition of web services in reliable manner. To implement reliability of web services, improved exception handling and compensation logic are added with this layer through our proposed framework architecture.

Our work focuses on the top of business process definition, particularly monitor and manages process execution to ensure process reliability and robustness. The higher layer is the service application layer which performs useful activities for users.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

B. The Reliable Service Composition Framework

Figure 3 gives a framework overview of Reliable Service Composition, which consists of four parts: - WS-BPEL Process Generator, Exception handling module, Evaluator module and Implementation module. From the top-down perspective, service designers initially get business specification requirements and they define normal business process in a graphical manner through WS-BPEL Process generator that facilitates for other people can easily understand the underlying business semantics. Nowadays, there are some nature tools available for this purpose, one of these is ActiveBPEL designer [8]. The exception handling module implements fault-handling mechanism that fully exploits WS-BPEL built-in exception handling. It gives a set of high level exception handling strategies. Whenever a fault occurs during runtime it first employs an appropriate exception handling strategies to repair it. If the fault has been fixed, the composite service continues its execution. Otherwise, exception handling strategy brings the composite service back to a termination state.

The Evaluator module is used to check the correctness of fault-handling logic. In particular, it identifies several occurrences of faults and their conditions must comply with it. If the fault-handling logic is verified as correct, then implementation module will transform these conditions into WS-BPEL codes and created them into the valid business process using WS-BPEL.

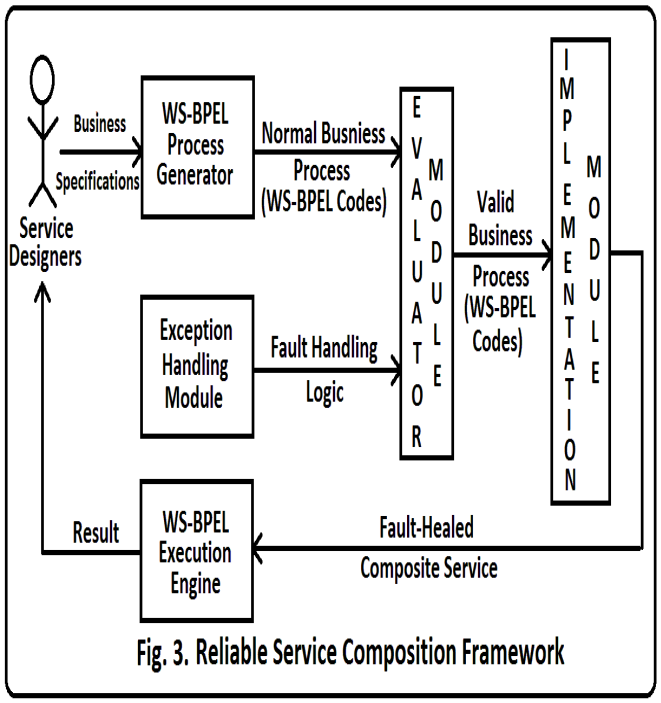

Fig. 3. Reliable Service Composition Framework

The implementation module is used to realize WS-BPEL constructs and complex exception handling functionalities expressed by exception handling module. The desired output of the implementation module is reliable composite services which can be deployed and executed on WS-BPEL execution engine.

V. FAULTS IN WEB SERVICES COMPOSITION

In this section, we identify the possible faults that can occur and their impacts during the composition of web services. A fault is a signal raised by a partner service towards the business process context. When an error state is reached to process context, in order to avoid abnormal termination of process, the fault handling strategy is called for recovering.

Here, we distinguish the three categories of faults [9]: Partner service faults, Process faults and System faults. Partner service faults are arising whenever there is the invocation to external web services. The primary reason is the web services commonly interact over unreliable Internet connections. This type of faults can be again classified into four subgroups: Unavailable Server faults, Incorrect parameter faults, SLA faults and Timeout faults.

- *Unavailable Server faults* are deliberately thrown by a partner service, due to failure to handle the BPEL process requests by server. For example, a flight service is invoked to book tickets, in order to complete this operation, the partner service have to access its database, but at this particular moment the database server is downed for some problem.
- *Incorrect parameter faults* are thrown when a service receiving incorrect arguments as input or a process sends a wrong request which is not usually processed by a partner service.
- *SLA (Service Level Agreement) faults* are thrown when partner service completes its operation but returned execution results cannot satisfy to the predefined SLA. For example, the expected completion time of one operation is 12 seconds in the SLA, but the actual completion time of the operation is 16 seconds.
- *Time-out faults* are thrown by BPEL process when a partner service fails to complete its execution within the predefined time frame, because of a slow network or an overloaded partner server may result in an elapsed delay in handling a request.
- *Process faults* are usually thrown by BPEL process itself. If BPEL designer whenever not initialized PartnerLink variables, then this type of faults can be thrown by process.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

For example, "Uninitialized Variable", is one of a common faults in BPEL which is thrown when the input variable for an invoke activity has not been initialized.

- *System faults* are raised from the supporting platform where BPEL processes are running. For example a temperature conversion web service is undeployed due to the change of business requirements. This is a situation arises when a service requestor invokes this service, the hosting server will inform it and that the service does not available on specific end point definition.

VI. WS-BPEL For Fault Handling

In this section, we explain the constructs available to provide fault handling methods which are already exits in WS-BPEL. There are four basic fault handling in BPEL generally involves [10]: scope, fault, termination and compensation.

A scope is a process container and context environment for other business activities and usually that can be denoted by a unique name. A scope activity provides handlers for faults, events and compensation.

A fault is a signal raised by a business process towards the enclosing scope, when an error reached in scope, the fault handled by corresponding fault handlers for fault recovering.

In an occasional case, a business process may need to explicitly signal a fault. For this situation, BPEL facilitates the throw activity. The faults thrown by throw activity must be handled in the BPEL business process. If fault is not handled properly then that will not be automatically propagated to the client. Rather, the BPEL process will terminate itself abnormally.

Termination is triggered when a running scope is stopped, since fault raised by a parallel process. So, termination mechanism used to recover from errors.

Compensation provides backward error recovery mechanism for a specified scope, which is explicitly invoked by the BPEL designer to undo the effect of a scope whose execution has already smoothly completed. There are two important types of constructs available in BPEL for providing fault tolerance. The first of these are compensation handlers, which are analogous to application specific rollback or cleanup for reattaining a state where execution can continue. The second type of constructs are fault handlers, they provide forward error recovery mechanism. These consists of catch blocks which are explicitly catch the thrown faults that are returned by invoke activity. Fault handlers are also attached to a scope that is attached with a group of activities.

But, a scope is to be terminated abnormally when a fault handler is invoked which is unlike a compensation handler. So that all activities in the scope that are abnormally terminated.

The aim of our work is to utilize these fault handlers as appropriate in order to implement various fault handling logic patterns.

VII. Attaching Fault Handling Logic To WS-BPEL

Our framework model facilitates to handle partner service faults. In this framework, service designers must mention the specification of fault handling logic during design time. The fault handling logic must conform to BPEL coding standards and according to the application requirements.

The identification fault is identified by adding event occurrence to the corresponding fault handler. The event is triggering for a fault and it named as a qualified fault name, which indicates that the fault occurs. The fault name is always associated with specific operation in WSDL document and namespace used as a prefix of the fault, so it can be identified as unique qualified name.

When a fault is identified, then action can take place to execute one of the exception handling strategies or consistent termination of the business process.

The most common requirements in the practical SOA applications [11], we present the following common exception handling strategies:

- *Retry*
  - Repeats the execution of service until it completes successfully.
  - It can be called when a web service is inaccessible due to network problems.
- *Ignore*
  - It does not take any special action but simply ignores it and allows the composite service to continue.
- *Wait*
  - It delays the invocation of a web service to a specified duration of time.
  - The reason for that some web services are available only in working time.
- *Alternate*
  - It selects another function equivalent service to perform the some task when a particular service fails.

International Journal of Emerging Technology and Advanced Engineering
Website: www.ijetae.com (ISSN 2250-2459 (Online), An ISO 9001:2008 Certified Journal, Volume 3, Special Issue 1, January 2013)
International Conference on Information Systems and Computing (ICISC-2013), INDIA.

The SLA and process faults normally triggered from normal business logic and they are often explicitly raised by throw activity in WS-BPEL, however they do not have any specific names in the WS-BPEL process level but they often have names that is, HTTP status codes at transport level.

VIII. Experimental Analysis

We have developed a prototype model that based on our framework. The initial part of the prototype is designed to specify the fault-handling logic in the form of event and its associated action patterns, which assists service designers. The next part of the prototype model is a set of Application Programming Interfaces (APIs) that realize the Evaluator module and implementation module. The final part of the prototype is WS-BPEL engine [12] that is used to execute composite web services. In this section, we conduct experimental study through the prototype and then we study the experimental results in terms of response time of composite services in WS-BPEL engine.

A. Implementation Setup

We first develop different web services through JAVA APIs such as JAX-WS, JAXB etc., and deploy them onto Glassfish server using Netbeans IDE 6.5. Based on these services, we develop a composite service called LoanApprovalAgent whose normal business logic is illustrated in Figure 1. Upon completion of above said process, we use our framework to develop FaultHandLAA, a fault-handling version of Loan ApprovalAgent. After that, we define event and its associated action patterns to represent fault-handling logic that is evaluated and implemented in WS-BPEL automatically by using our framework. Then, we deploy the composite services onto WS-BPEL engine.

To realize an unreliable environment, the credit check service is devised to complete successfully at a probability of 0.7 and other services are allowed to complete successfully always. All composite services are invoked 10 times. LoanApprovalAgent completes successfully in all tries. From the implementation setup, it is clearly understand that our framework can improve the reliability of composite services using fault-handling logic.

B. Performance Analysis

We first develop various basic web services and deploy them on set of PCs. After that, we develop a composite service whose business workflow logic is to invoke simple web services and deploy it on another PC.

In this setup all PCs have the same configuration. In on composite service, we develop four fault-handling logics, each of which uses one exception handling strategy to handle faults. Each composite service is executed 100 times and average response time is obtained. The response time for basic services is fixed (at 500 ms) to evaluate the experimental results and for comparison.

Chart 1 represents the average response time of composite services associating different exception handling strategies.

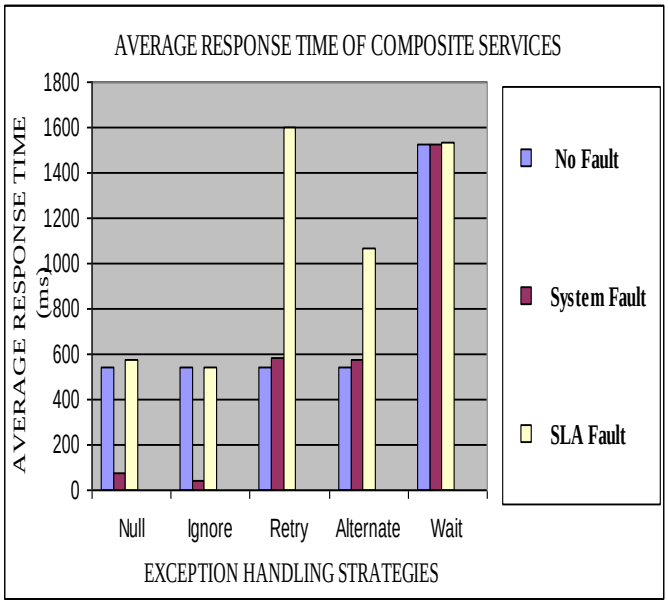

Chart 1 - Average Response Time of Composite Services

IX. Conclusions

In this paper, we have implemented framework architecture for fault-handling logic of reliable web service composition. We stated a set of high-level exception handling strategies and created a new classification of different faults. Based on above invention, we devised, exception handling logics that are associated with service composition techniques as the underpinning fault-handling mechanism for the framework. Moreover, we enhanced the framework by introducing GUI based module to help web service designers to specify and verify fault-handling logic and other module to automatically implement fault-handling in WS-BPEL. In overall, our framework model enriches the reliable fault handing of web services composition in efficient and easy manner.

To summarize, we hope that our framework model provides a effective solution to reliable fault handing in B2B collaboration using orchestration based web services composition.

REFERENCES

[1] F. Puhlmann, "Why do we actually need Pi-Calculus for Business Process Management," In Proc. of 9th International Conference on Business Information Systems (BIS 2006), Volume 85, pages 77-89.

[2] D. Jordan and J. Evdemon, "Web Services Business Process Execution Language Version 2.0, OASIS Standard," http://docs.oasis-open.org/wsbpel/2.0/serviceref, 2009.

[3] A. Avizienis, J. Laprie, B. Randell, and C. Landwehr, "Basic Concepts and Taxonomy of Depenable and Secure Computing," IEEE Trans. Dependable and Secure Computing, Vol.1, no.1, pp. 11-33, Jan-Mar. 2004.

[4] F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy, "Coordinated Forward Error Recovery for Composite Web Services," Proc. International symposium Reliable Distributed systems (SRDS'03), pp.167-176, 2003.

[5] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," IEEE Trans. Software Engg., Vol.26, no.10, pp.943-958, Oct. 2000.

[6] B. Benatallah, F. Casati, and F. Toumani, "Web service Conversation Modeling: A Cornerstone for E-business Automation," IEEE Internet Computing, Vol.8, no.1, pp. 46-54, Jan./Feb. 2004.

[7] C. Peltz,"Web Services Orchestration and Choreography," Computer, Vol. 36, no.10, pp.46-52, Oct. 2003.

[8] ActiveBPEL, http://www.active-endpoints.com, 2009.

[9] A. Liu, Q. Li, L. Huang, M. Xiao, "A Declarative Approach to Enhancing the Reliability of BPEL Processes," in Proc. ICWS'07, 2007, pp. 272-279.

[10] F. Montesi, C. Guidi, I. Lanese, G. Zavattaro, "Dynamic Fault Handling Mechanisms for Service Oriented Applications," in Proc. ECOWS'08, 2008, pp. 225-234.

[11] R. Hamadi, B. Benatallah, and B. Medjahed, "Self-Adapting Recovery Nets for Policy-Driven Exception Handling in Business Processes," Distributed and Parallel Databases, Vol.23, no. 1, pp.1-44, 2008.

[12] OASIS Web services Business Process Execution Language Version 2.0 ", http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf", April 2007.