A NOVEL APPROACH FOR EXCEPTION HANDLING IN SOA

Prachet Bhuyan[1] ,Tapas Kumar Choudhury[2] and Durga Prasad Mahapatra[3]

[1,2]School of Computer Engineering, KIIT University,
Bhubaneswar, Odisha, India
prachetbhuyan@gmail.com[1]
tkchoudhury24@gmail.com[2]
[3]Department of Computer Science and Engineering, NIT,
Rourkela, Odisha, India
durga@nitrkl.ac.in[3]

ABSTRACT

Services are certainly assuming an increasingly important role in modern application development, composite application. One may ask how to successfully implement Service-Oriented Architecture (SOA).The objective of the study to examine the key issue of the user's negative attitude towards introduction of SOA design. It is the fear of complexity that the SOA brings with its layers .Composite applications must be reliable and available, however it may appear more difficult to achieved, due to the multi-layered architecture of SOA. To reduce the fear of complexity, as well as to reduce the risk when introducing SOA architecture, it is necessary to use error handling methods in order to increase system fault tolerance. This paper looks at various error handling considerations associated with design of reusable services. It provides an outlineof what error handling considerations apply during SOA analysis and design phases. Also describes some best practices into designing these considerations. Finally ensuring that services are designed and implemented in all its completeness.

KEYWORDS

SOA, BPEL, ESB, Services, Choreography, Error Handling

1. INTRODUCTION

SOA is a business centric information technology architectural approach that promotes integrated and reusable business processes or services. In SOA, service is a fundamental element that can be independently developed and evolved over time. Each service is a self describing, composable, open software component. Business Process Execution Language for Web Services (BPEL4WS) was proposed for depicting interaction of web services in order to provide a process service. BPEL can compose various fine-grained services or business processes with different capabilities into a requested coarse-grained business processes. Service composition refers to the interoperation of autonomous and heterogeneous web services. BPEL provides an ideal way to composite services within SOA into complete business processes. However, web services usually communicate over internet connections that are not highly reliable. Web services can raise exceptions due to logical and execution errors [1]. BPEL uses provisions for exception handling and detecting failures, however, the inclusion of such provisions is a tedious assignment for the business process designer. Unlike in monolithic applications, error handling becomes a significant

DOI : 10.5121/csit.2012.2242

step in the design of SOA applications as SOA applications integrate heterogeneous IT systems across the organizational boundaries, vendor and partner IT assets. Focusing on error handling analysis early in the analysis and design phases ensures that appropriate error handling standards/guidelines are put in place for modules in different platforms. This paper identifies common error handling considerations such that architects and designers can address the issues while designing SOA Solutions.

1.1 Motivation

Business processes specified in BPEL will interact with partner processes through operation invocations on web services. Owing to web service distributed, heterogeneous and highly volatile nature, BPEL process is always inherently vulnerable to exceptions, such as connection error, may cause certain sub-process of composite services unavailable, obstructing thus the successful execution of the business process [3]. Web services can also raise exceptions due to logical and execution errors. During the execution of BPEL process, three kinds of exceptions: connection exception, logic exception and system exception may occur. Due to network instability, connection exception has not been rejected in the BPEL scenario and can only be detected by the execution environment such as connection refuses exception, serialization / deserialization error, service binding exception, response time-out exception and so on. Executing of an invoke activity in BPEL process may cause the connection exception. The programmer should catch the exception and add some common process such as retry, ignore to solve it. It is not only a duplicated work for the service invokers to write the repeat code, but also makes the BPEL process or web service client obscure and redundancy. The second category, named logic exception, includes those exceptions specific to the application logic of a web service. For example, an insufficient credit exception thrown by some loan approval service indicates that it is impossible for the custom applying for the loan payment to obtain the loan, because the credit limit has been exceeded. When the Insufficient Credit exception occurs, another web service or process such as logging, reducing the credit needs to be invoked either at the server side or the client side. However it is a kind of coupling and violates the principle of loose coupling in SOA.

1.2 Objective

The main objective of this paper is to:

- To propose a reliable exception handing technique in ESB layer.
- Adding more exceptions, handlings and rules into our system to complete the strategy. Besides, the system needs more web service and BPEL cases for the stress test, to verification the performance of the framework based on some special theory.
- Include the creation of a test environment and the development of appropriate fault handling mechanism in SOA.

1.3 Problem Definition

One of the important aspects of exception handling is propagating sufficient information to upstream nodes when an exception occurs. However, when an exception happens inside a
Web service, the details of the exception and contextual information is available only within the Web service. Web service specifications provide a SOAP DETAILS element in the SOAP FAULT structure to carry the exception details, but it is not mandatory for the service to populate the details. Also, the format/schema for carrying the exception details is not defined. This may lead to services populating the SOAP DETAILS element with their own custom format or ignoring this element completely [5]. So, service consumers either do not get the exception details or they get this information in different formats from different services. For example, an application using three services would have to have complex exception handling logic to deal with three different formats of exceptions. Web services do not have the capability of maintaining

stack trace information, which is very important for root cause analysis of any exception. The errors logged by services need to be traced back across each service node until the end consumer is reached to perform a root cause analysis. To solve these issues, common exceptions need to be converted to a standard, predefined exception message to promote consistency and prevent ambiguity to the service consumers. For example, HTTP errors like 404 Not Found, 401 Access Denied, 500 Internal Server Error, etc. can arise because of issues in accessing underlying services, even though the applications they are directly interacting with work without error. A user using a Web site may get an "Access Denied" error when he clicks on a button because an underlying service being used is denying access. This may confuse the user, as he might have been successfully authenticated by the application he is accessing.

2. LITERATURE SURVEY

Many attempts have been made for exception handling and analyzing in SOA. Huang T, et al. [2] used a stateful aspect extension method for monitoring web service at runtime. Wen Jiajia, et al. [7] provided Multi-Policy Exception Handling System (MPEHS) for exception handling in BPEL process. However their rules did not consider the reliability and extensibility of web service. Chen Liu, et al. [1] present a rule based approach to solve the exception handling problems (REHF). The approach takes the reliability and the extensibility of web services into consideration in the framework and improves the performance of the rule-based system while handling the exceptions. The rule repository and handling repository provide the interface extending rule and handling for the web service provider in the REHF. They provide a set of fundamental rules, exception categories and strategies. Stefan Bruning, Stephan Weileder and Miroslaw Malek introduce fault taxonomy for a systematic description of possible faults in SOA and show how they relate to each other [6]. This knowledge is essential for building dependable systems as well as for testing the system via fault injection. This fault taxonomy is complete in that it covers all typical steps of service interaction. However, the adaptations necessary for special domains cannot be covered here.

3. PROPOSED WORK-Error Handling

SOA analysis and design tasks are broadly classified into three major phases i.e. Service Identification, Service Specification and Service Realization as identified in Service Oriented Modeling and Architecture by Ali Arsanjani. Subsequent discussion of this topic is oriented around error handling considerations that apply to these three phases.

3.1 Error Handling during Service Identification

The goal of service identification is to come up with a candidate service portfolio that leads to identifying re-usable service portfolio. This phase involves analysis of business artifacts package that includes key requirements, business goals, capability models, Business Process Analysis Model (BPAM), use cases, etc.

3.1.1 Types of errors

Errors are broadly classified into two types [4]:

- **Recoverable Errors** - Recoverable errors are the errors that client programs can recover from to take appropriate alternate execution paths. Such errors are the result of failure to meet a particular business rule.
- **Non-Recoverable errors** - These are the errors that client programs cannot recover from. This kind of errors are result of some unexpected errors during runtime such as programming errors such null pointers, resources not available etc.

3.1.2 Identification of Business Errors

Analyzing through the business artifact package provides many opportunities to discover business errors associated with services. If there are existing asset(s) for a business service, those component interfaces could be used to discover additional business errors that are otherwise not identified in top down analysis. Business errors are what referred to as recoverable errors. Once the service portfolio is internal draft stages, evaluate the re-usable services for the following error handling considerations:

- **Business error scenarios**: Detailed description of condition that ags the business operation as invalid.
- **Error text**: Provides a brief description of the business error that service consumers will receive for a business error.
- **Error code**: Code that can be looked up for additional info about the error.
- **Suggestions**: Feedback to the service consumer such as examples of valid inputs, or displaying specific information related to the error etc.
- **Service area**: Identifies a service area that receives all notifications related to service system errors.

These attributes that define the business errors could either go into service contract or could be packaged into service response as needed.

3.1.3 Process failure recovery scenarios

- **Identify new operations** - Business process flows or any micro flows are to be analyzed in the light of business errors that individual services in a process flow could throw. Such an analysis could lead to discovering newer operations that are otherwise not found in a typical top down process decomposition tasks.
- **Updates to process models** - Service operation models/dependencies could be updated with the new operations discovered in the previous step.

3.2 Error Handling during Service Specification

Service Specification phase consists of tasks defining inputs and output messages, service and operation names, schemas, service composition, non-functional requirements and other service characteristics such as sync/async, invocation style, etc. for the services that are marked as to be exposed.

3.2.1 Characteristics related to Error Handling

Common service characteristics that are related to error handling are:

- Assured Delivery - Determine if a service requires assured delivery type of QOS. Such a requirement helps designers put in appropriate asynchronous messaging design patterns or use reliable messaging if implemented as web services.
- Monitoring requirements - Determine if the service business critical errors require being setup with proactive monitoring.
- Error mapping/transformation rules - Establish transformation rules for errors codes/info returned by the service provider and how it needs to be provided to service consumer. Having standard business error codes helps applications consume these services easily in terms of handling the service errors.
- Updated process flows - Existing process flows are to be updated with the newer operations or alternate execution paths as discovered in the identification step to handle business errors.

- Transaction attributes and boundaries - Nature of errors such as system Vs application errors influences how different runtime platforms handle automatic roll backs. Transaction attributes and boundaries in a process are to analyzed in the light of errors that can be expected from individual service invocations/transactions.

3.2.2 Common enterprise wide custom schemas

Identify metadata and common schemas to describe errors consistently across the enterprise. This data could include common attributes include date, time, error code, descriptions, severity level, message source, correlation id, etc. Thorough analysis of this metadata would turn out to be very useful for setting effective service monitoring.

3.3 Error Handling during Service Realization

Service realization phase is where the service model is mapped to service component and runtime/deployment model. This step typically involves designing service components, allocating the components to SOA stack layers choosing component interaction styles, runtime platforms and making architectural design decisions (ADD). Subsequent discussion of the subject will be focused around some best practices to implement error handling considerations in the three layers of typical enterprise SOA stack: business processes or choreography, mediation/BUS and component layers as highlighted in Figure 1.

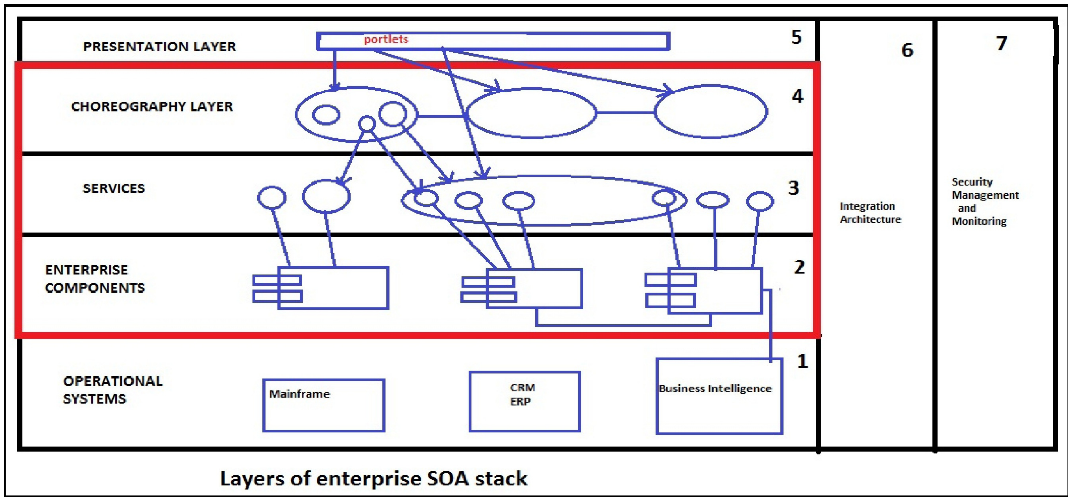

Figure 1: SOA Enterprise layers

3.3.1 Error handling in the business process/orchestration layer

Components deployed to this layer implementing business process flows or choreographies. The following error handling considerations apply here:

- **Fault Handlers** - Use of fault handlers is the most popular way of handling service errors returned from the service invocations initiated from within the orchestrations. Fault handlers are attached to specific tasks in a process flow or as a global fault handler for the entire process. When the process results in errors, fault handlers are invoked to implement the corrective tasks. Compensation transactions and manual rollbacks are configured with the fault handlers so that appropriate corrective actions could be applied to handle the process errors. Care should be taken not to use Fault Handlers for alternate execution paths instead should only be used to recover from the errors thrown in the process.

- **Service status info** - Choreography scenarios normally involve call more than one service. These service invokes from within the process could end up resulting in errors of different severity that could range from info, warning, error and fatal. It is a good practice to collect status description from each invoke such as return codes etc. into a repeatable array and return the same back to service consumer. Such a practice gives the ability to the service consumer to determine if the completion of the process involved any warnings/errors from some of the services that process invoked.

- **Threshold error severity levels** - Identify threshold error severity levels and design fault tolerance levels in service orchestration around these thresholds. Threshold levels could be set on any attribute or a combination of these that define the error, such as error severity levels, custom status codes etc. as opposed to solely relying on SOAP faults for determining process failures.

3.3.2 Error handling in the Services/Mediation/ESB layer

Enterprise Service Bus (ESB) layer is at the core of typical enterprise SOA stack (figure 2). This layer supports the transformation and routing capabilities required off of the enterprise reusable services. Components in this layer provide a well defined interface to the various provider implementations such as existing underlying assets and partner or vendor based services, by applying appropriate message and protocol transformations. Error handling by the mediation components mostly involves transforming the provider error structures into well defined error structures defined in the context of business domain. These components\ also could handle applying some complex transformation and mapping rules on the errors returned from the back end functional components to provide more simplified error info to the service consumers within the enterprise.

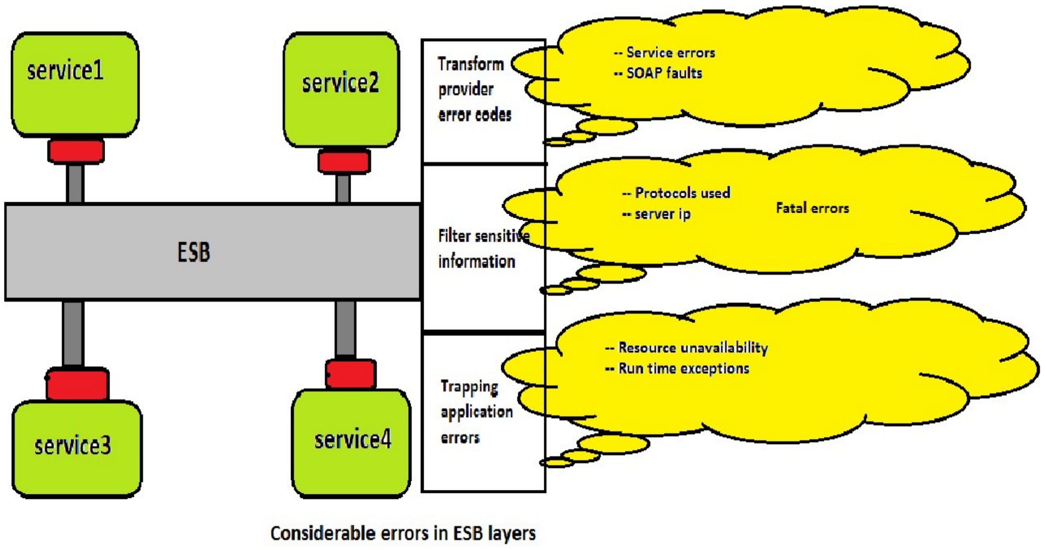
service1
service2
ESB
service3
service4
Transform provider error codes
Filter sensitive information
Trapping application errors
-- Service errors
-- SOAP faults
-- Protocols used
-- server ip
Fatal errors
-- Resource unavailability
-- Run time exceptions
Considerable errors in ESB layers

Figure 2: Considerable errors in ESB layer

- Transform provider error codes - It is possible that different service providers return service errors using different semantics. The range could involve anywhere from popular SOAP faults to very proprietary structures. Appropriate transformation rules can be applied here so that re-usable enterprise services return errors in a more consistent manner that enterprise applications could easily parse and implement appropriate handlers.

- Filter sensitive information- when internal service components throw fatal errors, the stack trace often contains sensitive information such as protocols used, server ips, etc. Appropriate filtering rules are to be established in this layer to filter any sensitive information in the stack trace. This strategy becomes all the more important when service responses are to be given out over the trusted networks.

- Trapping application errors - Any kind of technical errors experienced by the service components such as resource unavailability or some runtime exceptions etc. are to be transformed into a simple technical error messages. If native components did not log these errors, then mediation layers could pass all the stack trace info into logging but only return a generic text message back to the service consumer informing about temporary service unavailability.

A lot of error handling considerations mentioned for this layer is also possible to be implemented in the component layer. But there are number of ESBs and frameworks in the market that does these things in a lot more configurable and flexible manner than what individual platform developers could implement in their functional component implementations. Separation of such error handling mediation concerns to ESB layer relieves the platform developers from having to satisfy a variety of error handling consideration and have them focus more on implementing the business functionality resulting in greater developer productivity.

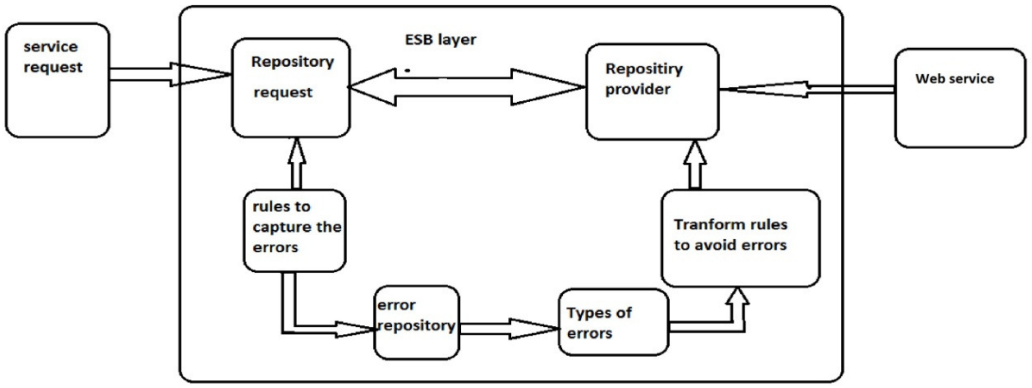

Figure 3: Error handling technique in ESB layer

3.3.3 Error flow steps

The following are the error handling steps in ESB layer (figure3).

- **Step 1**: When a service requesting for another web service the service request reach the request repository in the ESB layer.
- **Step 2**: request repository sends the address of the web service to the Repository provider.
- **Step 3**: Before it reach the repository provider the request repository sends the web address to the Rules to capture the errors.
- **Step 4**: If it finds any error then it sends the errors to error repository.
- **Step 5**: Error repository decides the error is in which type then it sends to the types of error.
- **Step 6**: Next the types of error send it to the transform rules to avoid error, here it applied some transformation then send it to the Repository provider.
- **Step 7**: Finally the repository provider searches the address of the web service and provide it to the service request.

3.3.4 Error handling in the component layer

Error handling by the components in this layer includes handling abnormal execution conditions such non-availability of a resource or some runtime conditions that the component is not programmed to handle or is considered in violation of logic. Components are required to handle such events to notify client programs and also do appropriate logging to help facilitate troubleshooting and service monitoring. In Java programming language, such events are thrown as exceptions and the API provides two different types of exceptions: checked and unchecked. Checked exceptions inherit from Exception class and are used to handle recoverable errors such as business error scenarios. Unchecked exceptions which are descendents of Runtime Exception class are the ideal candidate exceptions handle non-recoverable errors such as resource non-availability. The second part to component level error handling is to do appropriate logging. It is a good practice to perform logging closest to the source where the error occurred. When components throw application errors, they could log the exception at the appropriate interface within the component boundaries and then throw the exceptions. Use of correlation ids to identify the events and passing the same to calling applications would greatly enhance error tracking and monitoring by way of linking logs across different platforms.

4. COMPARISION WITH EXISTING APPROACH

Many SOA models have been proposed but error handling remains an issue. Focusing on error handling analysis early in the analysis and design phases ensures that appropriate error handling standards and guidelines are put in place for modules in different platforms. In this work we designed the error handling steps and how to handle the errors in the ESB layer which was not addressed earlier as far as our knowledge goes. Also identifying common error handling considerations that architects and designers need to address while going through the SOA solution design is still a challenge.

5. FUTURE WORKS

- Further elaborate Survey of SOA Error handling.
- Error handling in Service Identification, Service Specification and Service Realization.
- To propose a reliable exception handing technique using BPEL.

6. CONCLUSIONS

This paper provides SOA architects techniques to discover error handling requirements from the business artifacts package and how to analyze these while going through SOA analysis and design phase. Also provides some best practices to implement error handling in the three layers of SOA i.e. orchestration, mediation and component layers. A thorough upfront analysis of various error handling considerations help architects make the right decisions during design and implementation phases, platform and SOA stack products.

REFERENCES

[1] Chen Liu, Yanting Xu, D., and Xaiyu, L. “A rule-based exception handlings approach in SOA”. International Conference on Computer Application and System Modeling (2010).

[2] Huang T, Wu GQ, W. J.” Runtime monitoring composite web services through stateful aspect extension”. Journal of computer science and technology (2009).

[3] Kareliotis Christos, Dr. Vassilakis Costas, D. G. P. “Enhancing BPEL scenarios with dynamic relevance-based exception handling”. IEEE (2007).

[4] Poolla, H.” Error handling considerations in SOA analysis and design”. Enterprise Architecture (2010).

[5] Shukla, R. K. “Exception handling in service-oriented architecture”. HCL Technologies (2006).

[6] Stefan Bruning, S. W., and Malek, M. “A Fault taxonomy for service-oriented architecture”. IEEE (2007).

[7] Wen Jiajia, Chen Junliang, P. y. and Meng, X. "A multipolicy exception handling system for BPEL process". First International Conference on Communications and Networking in China (06, 2007).