Survey On Exception Handling In SOA

Vinod S. Patil
M.Tech student, Abha Gaikwad-Patil College of Engg, Nagpur, Maharashtra, India.

Pragati Patil
HOD, Department of M. Tech(CSE), Abha Gaikwad-Patil College of Engg, Nagpur, Maharashtra, India.

Parul Bhanarkar
Assistant Professor, Department of M.Tech(CSE), Abha Gaikwad-Patil College of Engg, Nagpur, Maharashtra, India.

ABSTRACT
Today, Service-Oriented Architecture (SOA) is a popular design paradigm for distributed systems. Services are performing an increasingly important role in modern application development and composite application. One may ask how to successfully implement SOA. The objective of the study to examine the key issues of the user's negative attitude towards introduction of SOA design. It is the fear of complexity that the Service-Oriented Architecture (SOA) brings with its layers. Most of the composite applications needed to 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, to reduce the risk as well as to generate light weight message usable by all types of clients (users) when introducing SOA architecture, it is necessary to use error handling and recovery methods in order to increase system fault tolerance This topic looks at various error handling considerations associated with design of reusable services. It provides a guideline about error handling considerations apply during SOA analysis and design phases.

Keywords
BPEL, Choreography, Error Handling, ESB, Services, SOA, SOAP, WSDL.

1. INTRODUCTION
Computer-based infrastructures are a necessity for many companies to handle their daily work today. The information system infrastructure of most companies is based on distributed systems, which consist of multiple independent computers connected by a network. A common design paradigm for distributed systems is Service-Oriented Architecture (SOA) [6].
The concept of service-oriented architecture (SOA) has been introduced for solving the problem of ensuring effective, reliable and secure interaction of complex distributed systems. SOA assumes that such systems are constructed from separate functional application modules (services) that have interfaces, defined by common rules (WSDL - description), and a dedicated invoke mechanism (SOAP messages).
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 arrange different fine-grained services or business processes with many 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. Just like in monolithic applications, error handling becomes a significant process 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.

2. REVIEW
Business processes specified in BPEL, which 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.

3. SOA STRUCTURE
The basic assumption of SOA is that there are many consumers that require services. In literature, consumers are also referred to as clients or customers. These terms are used interchangeably here. On the other side, there are many providers that provide services on the network. These two groups have to be linked together in a dynamic and adaptive way. This is usually done by a service broker [9], [10].
Service providers register their services at the broker (registry), service consumers request a service from the service broker, which returns a known provider for the requested service. Consumer and provider agree on the semantics. The consumer then binds himself to the service

provider and uses the service. The structure of this architecture is shown in Figure. 1.

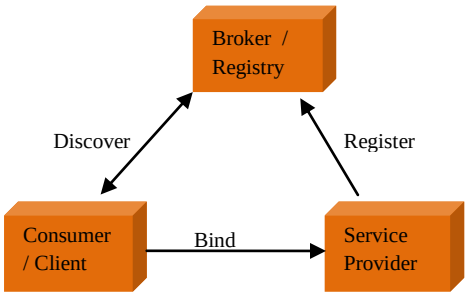

Figure 1: SOA Structure

3.1 SOA-specific Errors and Failures

In terms of fundamental concepts of dependability [11], threats for computer systems include errors, faults and failures. An errors are that parts of the system state that may cause a subsequent failure: a failure occurs when an error reaches the service interface and alters the service. A fault is the supposed or hypothesized cause of an error. All faults are gathered into three major fault classes for which defenses need to be devised: design faults, physical faults, interaction faults. We proceed from the assumption that most of the errors and failures occur during service binding and invocation, messages transferring and requests processing by web service. In this paper we specified different types of SOA-specific errors and failures (see Table 1).

Table 1: SOA-specific errors and failures

Sr. No	Type of error/failure	Error /failure domain
1	Error in Target Name Space	Client-side binding errors
2	Error in web service name	
3	Error in service port name	
4	Error in service operation's name	
5	Output parameter type mismatch	
6	Input parameter type mismatch	
7	Error in name of input parameter	
8	Mismatching of number of input service parameters	
9	Web service style mismatching	
10	Suspension of web service during transaction (getting into a loop)	Service errors and failures
11	System error during processing (like "Divide by Zero")	
12	Calculation error during processing(like, "Operand Type Mismatch	
13	Application error raising user exception(defined by developer)	
14	Network connection break-off	Network and system failures
15	Domain Name System (DNS) is down	
16	Loss of packet with client request or service response	
17	Host unavailable (off-line)	
18	Application Server is down	

4. ERROR HANDLING

Unlike in monolithic applications, error handling becomes a significant 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 topic identifies common error handling considerations that architects and designers need to address while going through the SOA solution design. 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.

4.1. 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 [4]. This phase involves analysis of business artifacts package that includes key requirements, business goals, capability models, Business Process Analysis Model (BPAM), use cases, etc.

4.1.1 Types of errors

Errors are broadly classified into two types:

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.

4.1.2 Identification of Business Errors

Analyzing through the business artifact package provides many opportunities to discover business errors associated with services [4]. 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 flags 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.

4.1.3 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.

4.2 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 [4].

4.2.1 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.

4.2.2 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.

4.3 Error Handling during Service Realization

Service realization phase is where the service model is mapped to service component and runtime /deployment model [4]. 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 2.

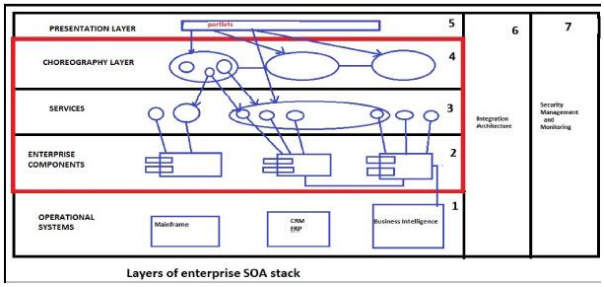

Figure 2: SOA Enterprise Layer [12]

4.3.1 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 [4]:

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.

4.3.2 Error handling in the Services/Mediation/ESB layer

Enterprise Service Bus (ESB) layer is at the core of typical enterprise SOA stack (figure 3). 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.

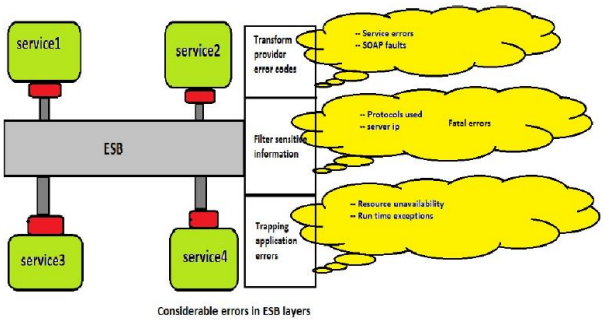

Figure 3: Considerable errors in EBS layer [12]

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.

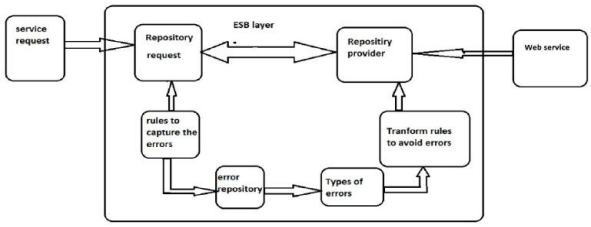

Figure 4: Error handling technique in ESB layer [12]

4.3.3 Error flow steps

The following are the error handling steps in ESB layer (figure3) [12].

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.

4.3.4 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 [4]. 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 nonavailability. 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.

5. 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.

6. 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).
[8] L. Srinivasan and J. Treadwell. An overview of service-oriented architecture, web services and grid computing. HP Software Global Business Unit, 2, 2005.
[9] W3C, "Web services architecture," Febuary 2004. [Online]. Available: http://www.w3.org/TR/ws-arch/
[10] G. Denaro, M. Pezz´e, D. Tosi, and D. Schilling, "Towards self-adaptive service-oriented architectures," in TAV-WEB '06: Proceedings of the 2006 workshop on Testing, analysis, and verification of web services and applications. New York, NY, USA: ACM Press, 2006, pp. 10–16.
[11] Avizienis A., Laprie J.-C., Randell B., Landwehr C. "Basic Concepts and Taxonomy of Dependable and Secure Computing", IEEE Trans. on Dependable and Secure Computing. Vol.1, № 1. – P. 11-33, 2002.
[12] Prachet Bhuyan ,Tapas Kumar Choudhury and Durga Prasad Mahapatra, "A Novel Approach For Exception Handling In SOA", David C. Wyld, et al. (Eds): CCSEA, SEA, CLOUD, DKMP, CS & IT 05, pp. 425–433, 2012.