doi: 10.20532/cit.2017.1003569

Extended Fault Taxonomy of SOA-Based Systems

Guru Prasad Bhandari[1] and Ratneshwer Gupta[2]

[1]DST-CIMS, Institute of Science, Banaras Hindu University, Varanasi, India
[2]School of Computer and Systems Sciences, JNU, New Delhi, India

Service Oriented Architecture (SOA) is considered as a standard for enterprise software development. The main characteristics of SOA are dynamic discovery and composition of software services in a heterogeneous environment. These properties pose newer challenges in fault management of SOA-based systems (SBS). A proper understanding of different faults in an SBS is very necessary for effective fault handling. A comprehensive three-fold fault taxonomy is presented here that covers distributed, SOA specific and non-functional faults in a holistic manner. A comprehensive fault taxonomy is a key starting point for providing techniques and methods for accessing the quality of a given system. In this paper, an attempt has been made to outline several SBSs faults into a well-structured taxonomy that may assist developers in planing suitable fault repairing strategies. Some commonly emphasized fault recovery strategies are also discussed. Some challenges that may occur during fault handling of SBSs are also mentioned.

ACM CCS (2012) Classification: Applied computing → Enterprise computing → Service-oriented architectures

Information systems → World Wide Web → Web services

Software and its engineering → Software organization and properties → Extra-functional properties → Software fault tolerance

Keywords: fault, SOA-based System (SBS), Service Oriented Architecture, SOA, distributed system, QoS, non-functional faults

1. Introduction

SOA (Service-Oriented Architecture) is a popular distributed design paradigm that provides architectural style to enable applications to be built using service as a key element. The system developed using the concept of SOA is known as SOA-based System (SBS). The main property of SOA is to dynamically discover services from different service providers and their composition at runtime in order to construct the software system. This transparency makes the SOA effective but also brings the possibility of various faults to occur at different stages.

In technological terms, a fault is an abnormal condition of the system (or in a component, equipment, or sub-system) which may lead to a failure. In other words, a fault is a problem that occurs when a service invocation made by an SBS results in some abnormal behavior at runtime [1]. IEEE [2] defines more precisely a software fault as an incorrect step, process or data definition in a computer program. The error is a human action that generates an incorrect result. Failure is the inability of a system or component to accomplish its required functions within specified performance requirements. The faults can easily escape the attention and increase in their severity. If a fault is not handled properly, then it increases the system failure rate. Detecting a faulty service is a very difficult task. The fault can only be detected in the execution step when the service is actually executed. The fault analysis process takes fault data as input and determines a suitable remedial strategy for the fault instance [3].

SBS must be capable to manage faults, i.e. a system can detect a fault and its root cause and recover it from fault situation. A fault leads to a failure when the system does not perform up to the specifications and shows the behavior that is a visible deviation from the expected behavior of the system. So, immediate recovery action is needed to handle the fault and to keep the system free from failure to increase its dependability. Correct execution of its functions without any interruptions ensures the dependability of a system. To increase the dependability of the SBS, a fault management should be performed that covers the following basic operations: 1) detection of faults in SBS by identifying the location of the fault, 2) diagnosis of the fault; the root cause of the fault is identified and, finally 3) recovery from the fault situation, applying appropriate recovery strategy and back to its smooth functioning.

A taxonomy is a practicable idea for understanding similarities and differences of the methods and techniques based on their characteristics. The main contribution of this paper is a fault taxonomy of SBS. The attributes of a fault can be classified including its severity, failure type, time of failure occurrence and the type of fault. Different aspects of faults in SBS have been studied and outlined in three categories as SOA life cycle-specific faults, distributed system faults, and non-functional faults. Some commonly emphasized fault recovery strategies are also discussed. Some challenges that may occur during fault handling of SBSs are also mentioned.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 briefly introduces Service Oriented Architecture (SOA). Some faults in SBS, observed in the literature, are summarized in Section 4. The proposed extended fault taxonomy of SBS and a brief explanation of every identified fault are mentioned in Section 5 and the following subsections. In Section 6, some commonly adopted fault recovery strategies for SBS have been discussed. Section 7 points out some SBS fault handling challenges. Finally, Section 8 summarizes the work with the concluding remarks.

2. Related Work

In the literature, most of the taxonomies have underlined the general software faults. Some efforts are centered on distributed system and Component-based System (CBS). Very few research efforts are available related to fault taxonomy of SOA based system (SBS). A study of Stefan *et al.* [4] is very close to our work. They have categorized SOA fault taxonomy into five types such as publishing fault, discovery fault, composition fault, binding fault and execution fault corresponding to SOA life cycle stages. Their presentation of fault taxonomy of SOA is unaware of the interactions among the faults. They have presented a well-defined collection of SOA related faults, nevertheless they have weakly mentioned how a fault propagates to another fault. Cheun *et al.* [5] have defined fault taxonomy by extracting all target elements and inter-relationships among the elements for service fault management and presented prototype implementation using cause taxonomy and checked its validity experimentally.

Avižienis *et al.* [6] presented a comprehensive paper on taxonomy of the dependable and secure computing. Mariani [7] has proposed a fault taxonomy related to the component-based systems along with the brief explanation of the causes and consequences of faults. In his approach, faults are categorized as syntactic faults, semantics faults, non-functional faults, connectors faults, topology faults, and other faults. There are some differences between CBS and web services. Web services execute remotely, whereas components are mostly downloaded to execute locally on the client and they have to deal with considerable heterogeneity in platforms middleware. Hummer *et al.* [8] have also used well-established fault dimensions proposed by Avizienis *et al.* [6], to elaborate a fault taxonomy for Event-based Systems (EBS) by discussing fault instances across the five sub-areas of event processing. Chan *et al.* [9] have presented a fault taxonomy based upon [10] for web service composition that covers physical faults, development faults, and interaction faults.

Fault taxonomy, classifying only security faults of software with its application is presented by Aslam *et al.* [11]. They have categorized security faults as synchronization errors, condition validation errors, configuration errors and environment faults. Their classification scheme is helpful in the understanding of computer security faults that cause security breaches. Vijayaraghavan *et al.* [12] have presented bug taxonomies with some bugs and challenges in the real software environment examples. Kidwell *et al.* [13] have mentioned that fault classification provides vital information for software analytics and that machine learning techniques like clustering can be applied to learn a project-specific fault taxonomy.

From the above literature review, it can be observed that several efforts are available regarding the fault taxonomy, but there are limited efforts available regarding fault taxonomy of SOA based systems. We extend the above contributions further by presenting a fault taxonomy, especially meant for SBS, that covers distributed, SOA specific and non-functional aspects together. The proposed taxonomy tries to collocate possible faults of SBSs in well-structured classification on the basis of their severity, types and time of occurrence.

3. Service Oriented Architecture

According to IBM definition [14], "SOA" (Service Oriented Architecture) is a set of architectural principles, patterns, and criteria, that address characteristics such as modularity, encapsulation, loose coupling, separation of concerns, reuse and composability. Microsoft defines it as "a loosely-coupled architecture designed to meet the business needs of the organization" [15]. Mainly there are three parties in the SOA-based System (SBS). These are service provider, service broker (or service repository or service registry) and service consumer. The service provider constructs a service or a set of services and registers them into service repository. The service provider creates a web service and deploys it into the service repository. Service broker makes the information of the available web service to the service consumer. Service requester or service consumer demands for a service or a set of services according to the need. Service providers and service consumers are loosely coupled. They communicate with each other through a service broker. A service provider can also be a service consumer. Figure 1 illustrates a basic SOA interaction structure. In further subsection, SOA life cycle steps are briefly described.

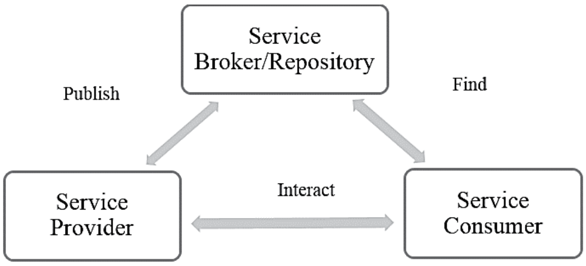


Figure 1. SOA interaction structure.

3.1. SOA Life Cycle

Following are the different stages of SOA life cycle.

Publishing. Service providers construct services on the network and provide their corresponding service descriptions. All web services are described by WSDL (Web Service Description Language) documents.

Discovery. If a service consumer needs an appropriate service to perform a certain task, then he has to discover a corresponding service among all the service providers. A comparison between a required service and a service in the repository can be made, based on the search criteria, to discover the suitable one.

Composition. If there is no such an identical service discovered in the service depository then there is still another possible option to compose two or more services to fulfill the service consumer's requirement. Two or more services can dynamically be composed at runtime, either by the service choreography or service orchestration.

Binding. At this stage, desired service execution permission is granted to the service consumer after applying authentication, authorization, and accounting. If a service is bound to the consumer system, then it can be used to fulfill the requirement. SLA (Service Level Agreement) describes the details of the agreement between service consumer and service broker.

Execution. If the service is successfully bound to a consumer, service(s) can be executed. All the input parameters are transferred to the service provider system and output parameters are returned to the consumer.

4. Some Observed Faults in SOA Based Systems

In this section, some of the observable SBS faults, available in the literature are presented. Basically, a fault is a problem that results in some abnormality condition at runtime [1]. Service faults can fall into one of the four categories: healthy, impacted, hidden and faulty [16]. From a temporal perspective, Huang *et al.* [16] have mentioned four parameters of a service s_i; service execution time $E(s_i)$, accumulated execution time $A(s_i)$, execution time threshold $T(s_i)$, and intermediate deadline $D(s_i)$. Among the four parameters, $E(s_i)$ and $A(s_i)$ are monitored at runtime. Based on these four values, a service falls into one of the four categories:

- *Healthy*: Formally, if and only if $E(s_i) \leq T(s_i)$ and $A(s_i) \leq D(s_i)$, which means it is in a good situation. *Healthy service* does not show any abnormal behavior at all, but if it depends on other faulty service or generates fault itself, then it can be faulty. It depends upon the nature of the fault and on what kind of behavior it possesses.
- *Impacted*: If and only if $E(s_i) \leq T(s_i)$ and $A(s_i) > D(s_i)$, which means s_i is not the root cause to originate fault, but its QoS is impacted by other service say s_j. If the workflow goes through the impacted region, the fault may be propagated to the system.
- *Hidden*: Hidden fault can be defined formally as, if and only if $E(s_i) > T(s_i)$ and $A(s_i) \leq D(s_i)$, which means it should be reconfigured to avoid any problem caused by inter-dependency. The *hidden fault* is problematic to identify and takes any remedial action for redemption.
- *Faulty*: Faulty service shows abnormal behavior at runtime. Formally, it can be defined as, if and only if $E(s_i) > T(s_i)$ and $A(s_i) > D(s_i)$, refers it is a root cause of the application-level violation.

In this review, we highlight the faulty cases in SBSs. There are so many reasons for the occurrence of faults in SOA. Service mismatch is one of the major causes of the fault. A service mismatch occurs during development time, if not corrected, creates fault at runtime. Service mismatch is the problem that may occur when a service does not fully match the feature expected. If a fault is active, then it generates an error. Fault can be either initiated by external interaction or by internal dormant fault [6]. If error propagates in an SBS, then it causes failure, of the system resulting in incorrect service. The creation of fault and its propagation is shown in Figure 2.

A byzantine fault is a SBS fault highly emphasized by the researchers, it presents different symptoms to different observers. A system can lose the execution due to byzantine failure which is created by root cause of a byzantine fault. Zhao *et al.* [17] have proposed a framework for byzantine failure tolerance messaging framework (BTF-WS) which is based on Castro and Liskov's BFT algorithm to maximize the interoperability. This framework has a drawback as it needs high cost in terms of processing power because every client request effectively processes twice to maintain the replicas for the security purpose.

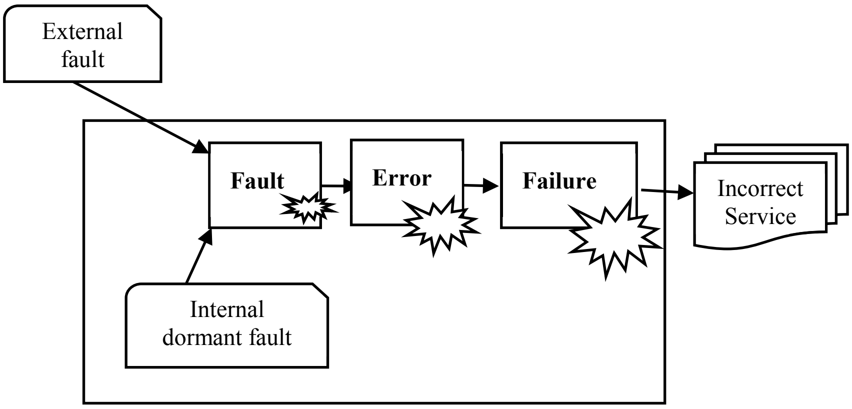


Figure 2. Propagation model of fault, error and failure.

Wang *et al.* [3] have categorized fault types on the basis of four contexts such as: functional context, QoS context, domain context and platform context. Belli *et al.* [18] have shown fault taxonomy on the basis of sequencing pattern of the graph-based approach; positive sequencing faults, negative sequencing faults. Balbastro *et al.* [19] have focused on the error during service delivery as a latent error and dormant fault. Zhai *et al.* [20] have classified SOA faults on the basis of stages of SOA life cycle.

Friedrich *et al.* [21] have discussed permanent fault and transient fault. The permanent fault occurs due to the faulty operation and may result in faulty behavior. There may be a transient fault or a temporary fault, although the subsequent operations of this operation are correct. Jinfu *et al.* [22] have identified three types of faults: 1) interaction faults of several parameters of the same service, 2) interaction faults of parameters for different services and 3) vulnerability faults. Through effective testing, one can find the interaction fault [23], [22], [24] between parameters of the same service or inter-service fault between two services.

Ye *et al.* [23] have categorized faults into two major types: type 1 relates to the internal fault to a service and type 2 corresponds to inconsistency faults that cut across different services in a service composition. Different faulty versions of service compositions can be obtained by seeding one fault to every original service compositions using mutation testing techniques. Each faulty version has one mutation.

In Table 1, we summarize different types of fault in SBS observed in the literature. The

Table 1. Fault types, fault tolerant systems and assumptions.

Fault	Fault Tolerant approach	Fault assumption/Cause	Remarks
Prescribed policy violation fault [25], [26]	Logging based approach	Opening socket connections, reading and writing files, and accessing critical memory regions.	– History based approach to analyzing whether prescribed policies are violated or not. – Long-lived transactions can hold locks on external objects for long periods of time.
Transaction isolation fault [19]	Synchronized transactional fault tolerant approach	Transaction evolving from a flat transaction, transactions with savepoints, nested transactions	– A mechanism for handling exceptions in a synchronized way – FT (Fault Tree) technique for providing correct service in case of long-lived transaction
Service unavailability fault [27], [24]	Modelling approach	– Temporal unavailability of service = Estimated execution time < end available time – Temporal service unavailability	– Analysis impact region and applying repair strategies: replacement, re-composition, and renegotiation – Calculates temporal negative impact, inconsistent and satisfied OR consistent and unsatisfied OR inconsistent and unsatisfied.
Byzantine fault (arbitrary fault) [17]	Byzantine fault-tolerance framework (BT-F-WS)	Byzantine fault: Different symptoms to different observers.	– To achieve maximum interoperability – Implementation in Standard SOAP messaging framework – On a testbed consisting of 20 Dell SC440 Servers connected by a 100 Mbps Ethernet on SUSE Linux
Network traffic fault [28]	Prototype-DRTS (Distributed Real-time Systems) tool	Availability of network and nodes congestion, transmission errors and delays	– Result: identify constraint violations, increase the probability of exhibiting network traffic related faults. – Uses UML 2.0 model based on analysis of control flow in sequence diagrams
Timeout exception [29]	Formal approach	WS-CDL and WS-BPEL supportive exception handling – Interaction failures – Timeout errors – Validation errors	– In case of deadline overrun, the event handler will take over and halt the process. – Using a document ordering and delivery process

Table 1. (cont.).

Transient fault [30]	Framework for fault-tolerant composition	Assume there is a network problem	– The transient fault causes unavailable service
SLA Claim fault [31]	Theoretical model	Customers making false or repetitive claims	– Customer's malicious or aggressive Internet activities do not guarantee SLA claims and violate the AUP (Acceptable Use Policy) – False or repetitive claims are also a violation of SLA.
Latent errors and dormant faults [19]	CAA-DRIP framework	Error at service delivery	– Simulation technique
Adaption faults [32]	Context-Aware Adaptive Applications (CAAAs)	– Faults related to architectural layering and context-based – Unable to change according to surrounding	– Architecture choices must inform and be informed by validation and verification techniques – Focus on different faults that are realized as failures in higher layers.
Vulnerability faults (interaction faults) [22]	A fuzzy mutation approach algorithm	Interaction faults of parameters within services and inter-services; testing approach	– Extract URL information of the Web services to obtain the interface information. – Testing involves injecting only one mutant at a time. – Implemented in c#, the efficiency of the proposed system is 54%.
SLA violation [33], [34]	SLA violation handling approach using incremental time impact analysis	– Time inconsistency and unsatisfactory condition. – Change of service due to the service replacement.	– Find impact region, existing impact region and expand the impact region and increase the range if it does not produce robust and adaptive SBS. – Recover and handle the violation in relation to the strategy of minimizing the no. of service change.
Crash fault [35]	Byzantine fault tolerance model	– Crash faults and malicious fault – Crash fault due to hardware failure or malicious fault due to software malfunction	– The server is replicated to 4 replicas to tolerate from fault replica. – Optimistic replication technique is used. – Reduce 20% of the peak system throughput to keep 4 replicas
Cascading failure [36]	Cascading Failure Tolerance	Failure in one node (service) depending on another service	– Increasing the number of alternate services can significantly improve the network tolerance if each service has only few alternate services available. – Scale-free topology has been used.
Temporal violation [37]	Temporal violation handling point-selection strategy	Failures of system on-time completion due to uncertain system performance, failure of timely completion of workflow activities	– Throughput consistency state verification – Violation handling point selection – The mean time response time of workflow instances is 79.64 s, and mean response time delay of postponed workflow instances is less than 7.9 s
Composite service unavailability [38]	Protocol-based automatic failure recovery	Runtime unavailability of component services that results in composition failure.	– Migrating the failed execution into a best alternative execution of the composite service. – Computing the number of invisibly compensated transitions is NP-complete. – The finite state machine to model the approach.

data in the table are organized in the following manner. Firstly, we mention the types of fault, their corresponding fault tolerance approach, the possible reason for that particular fault and finally some special remarks. We try to cover various faults that may occur in an SBS.

5. The Proposed Extended Fault Taxonomy of SOA-based System

The proposed fault taxonomy is based on several studies [3], [4], [39], [30], [18] and [9]. We have categorized SBSs fault into three classes in our proposed fault taxonomy of SBS: SOA cycle-specific fault, distributed system related faults and non-functional faults. The proposed classification scheme can assist in the proper understanding of faults that results in security breaches by categorizing faults and grouping faults that share mutual characteristics. Faults may overlap each other as one fault may become the cause of another fault.

SOA life cycle has 5 major steps (as mentioned in Section 3). Each step corresponds to unique faults; publishing fault, discovery fault, composition fault, binding fault, and execution fault. The categorization of SOA specific faults is motivated by the fault taxonomy of Brüning [4]. However, their taxonomy fails to represent the cascading faults in the SBS. SBS belongs to the distributed system, thus we have made one class for distributed system faults that covers the hardware faults, software faults, communication errors, and user wrongs. All other faults dealing with non-functional properties are categorized into the third category called 'non-functional faults'. Figure 3 depicts the proposed fault taxonomy framework and Figure 4 displays the proposed fault taxonomy of SBS. In the following subsections, the proposed fault taxonomy has been explored.

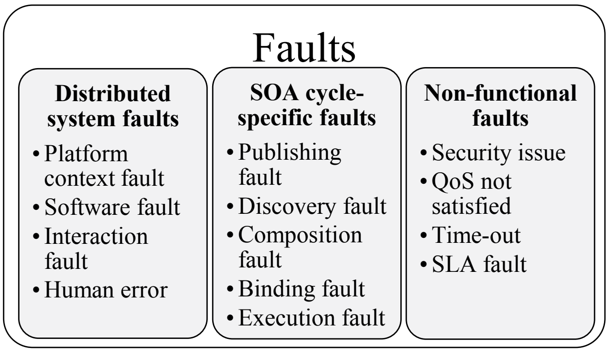


Figure 3. A proposed extended fault taxonomy framework.

5.1. Distributed System Faults

SOA is a distributed system, so faults occur in a distributed system inherently occur in an SBS as well. Mariani [7] suggests a fault taxonomy that narrates distributed system faults, but their taxonomy is targeted for a component-based system. Data corruption, hanging processes, misleading return values. Misbehaving participating machines, hardware/software/network aging are the major causes generating a fault in SBS [40]. Intermittent Internet outages, outages caused by hardware (server/node) crashes and downtime due to the maintenance of hardware and software upgrades and bug fixes may generate distributed system faults [40]. Moreover, resources exploitation, insufficient disk space during data reading and writing to the disk cause temporary failures of all involved computational jobs. We have sub-categorized distributed faults into four classes platform context fault, software fault, interaction fault and human error/wrongs. Each of them is individually introduced as follows.

5.1.1. Platform Context Fault

Adaption faults, hardware or device related faults and connectivity related faults are acknowledged as platform context faults. Due to different platforms and the environments, the new technology cannot adapt to the emerging technology. Hardware change, and communication medium change bring the system into the unintended state. Since SBS is a platform independent system, the manufactured hardware platform should cope with the software technology. We have investigated a few platform context faults related to SBS. Following subsections describe each of them briefly.

Adaption fault. Architecture choices must inform and be informed by validation and verification techniques in order to mitigate the impact of adaptation faults and their associated failures. Different faults are realized as failures in higher layers. Faults tend to be detected in layers other than the ones in which they occur. Faults related to architectural layering and context-based system are called adaption faults. Adaptation faults are unable to change according to surrounding environment-speed/location. Sama *et al.* [32] have discussed adaption faults and proposed Context-Aware Adaptive Applications (CAAAs) system as fault tolerant system.

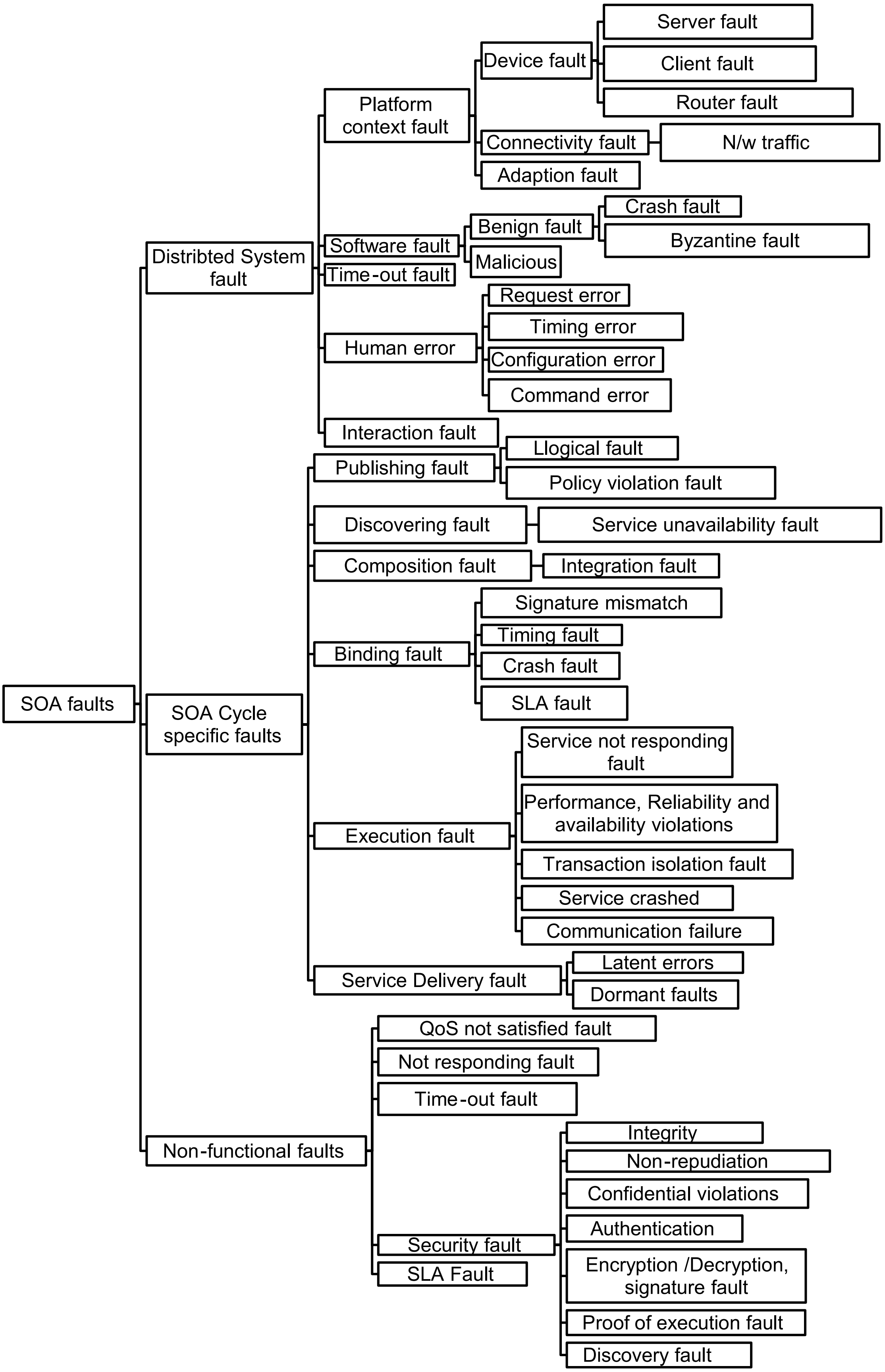


Figure 4. A proposed extended fault taxonomy for SBS.

Device fault. Hardware components can be faulty at any stage of process execution. Hardware glitches, power failure, and technical failure may occur in any component of the system that brings the distributed system failure. We have noticed three kinds of faults: server fault, client fault, and router fault as device fault or node fault. Faults in a particular node or communication channel, transmission media, memory, processing units, storage media, surges and spikes of power supply are possibly observed device faults or crash faults. Hardware faults causing subsequent software faults are physical faults with their origin in the physical device [41].

Connectivity fault. Congestion, transmission errors, transmission delays are the connectivity related problems that cause unavailability of network and nodes. Garousi *et al.* [28] have implemented prototype-DRTS (Distributed Real-Time Systems), a prototype tool that uses UML 2.0 model based on the analysis of control flow in sequence diagrams. As a result, their tool is able to identify constraint violations and increases the probability of exhibiting network traffic related faults. Liu *et al.* [30] have stated that transient fault may cause unavailability of service due to network problem in service-oriented architecture.

5.1.2. Software Fault

In this category, we have presented mainly two types of faults, such as *byzantine fault* and *malicious fault*. The term *byzantine fault* was coined by Lamport to represent an arbitrary fault, which might be a crash fault due to hardware failures or a malicious fault due to software malfunction caused by an intrusion into the system. *Byzantine fault* often refers to arbitrary fault and shows different symptoms to different observers. *Byzantine fault* tolerance refers to the capability of a system to provide correct services to its clients in the presence of byzantine faults. A study [42] by Chai *et al.* has proposed a byzantine fault tolerance approach using state-machine replication with byzantine agreement algorithm. To ensure the SBS is secure against malicious attacks we need to analyze and understand the characteristics of faults that can subvert security mechanisms. Zhao *et al.* [17] have also developed byzantine fault-tolerance framework named BTF-WS to achieve maximum interoperability. They have implemented their BTF-WS in standard SOAP messaging framework and tested it on a testbed consisting of 20 Dell SC440 servers connected by a 100 Mbps Ethernet on SUSE Linux.

The malicious fault may occur due to software malfunction caused by an intrusion into the system. Chai *et al.* [35] have also discussed byzantine fault (crash fault and malicious fault) and modeled byzantine fault tolerance system. Their byzantine fault tolerance mechanisms guarantee correctness of properties. The server is replicated to 4 replicas to tolerate from fault replica. The optimistic replication technique has been used. It reduces 20% of the peak system throughput to keep 4 replicas.

5.1.3. Interaction Fault

URL (Unified Resource Locator) information of the Web services it used to obtain the interface information. It can parse WSDL documents to obtain the SOAP messages. Vulnerability faults cannot be detected if the testing involves injecting only one mutant at a time. Chen *et al.* [22] have proposed a fuzzy mutation approach algorithm to reduce the interaction faults/vulnerability faults assuming that there are interaction faults of parameters within services and interservices. From the implementation in C#, 54% efficiency of the proposed system has been achieved. Failure in one node (service) may affect another service. Increasing the number of alternate services can significantly improve the network tolerance if each service has only a few alternate services available. Dependency which creates the failure of one service can cause the failure of another dependent service. Effect of network topology on tolerance is more significant on a lower degree of an alternative. Scale-free topology has generally the highest tolerance. Lhaksman *et al.* [36] have proposed cascading failure tolerance system to deal with cascading failure caused by service interaction.

5.1.4. Human Error

Human error causes the service interruptions on the smooth execution of SBS and creates a problem on service delivery. Human error is difficult to model and prevent. Shwartz *et al.* [43] have noticed four human wrongs: request, time, configuration item and command wrongs. The user or the operator of the SBS may send the request with invalid input or logical error. In that case, the system cannot function properly. The user can send a request at an invalid time like ordering an item after a time-out. If the SBS system developer has not properly handled the exception of the system as many ways the system can be configured then the system may produce the invalid configuration result. A very common mistake by the user is an invalid command resulting from syntactical or symmetrical error in the command.

5.2. SOA Cycle-specific Faults

SOA cycle-specific faults apply to SOA only. Some distributed system faults are also common with SOA cycle-specific faults because SOA is also a distributed design paradigm. We have also adopted some aspects of the fault taxonomy of Brüning *et al.* [4]. We try to include some additional SOA cycle-specific faults in addition to Brüning *et al.* taxonomy. There are five steps of SOA cycle: publishing, discovering, composition, binding and execution, as mentioned in Section 3. We have categorized SOA cycle-specific faults into five categories as described in the following subsections.

5.2.1. Publishing Fault

The fault which occurs in the publishing stage of SBS is called publishing fault. Logical faults and policy violations are noticed as publishing fault in our proposed fault taxonomy of SBS. Bartole *et al.* [26] have discussed fault regarding prescribed policy violation. They have also proposed a history-based approach to analyzing whether prescribed policies are violated or not. Their approach deals with security-relevant activities, for example, opening socket problem, a problem on reading and writing files, and problem on accessing critical memory regions etc.

5.2.2. Discovering Fault

Service unavailability fault has gained more attention from the researchers. Ismail *et al.* [27] and Wang *et al.* [24] have presented mathematical model about the temporal unavailability of services. They have stated that a service can be unavailable for certain period of time due to its maintenance, or some services may be available for only a certain time, for example, railways ticket booking service, movie theatre ticket booking service. Temporal unavailability of service can be calculated by estimating execution time and end available time.

5.2.3. Composition Fault

Service composition process may fail at intermediate state due to an incomplete description of goal service requirements or due to the fact that the user is unaware or uninformed of the functionality provided by the existing participant services [44], [45]. For example: if the *"Online Booking"* service is lacking all other committed services such as *"Item Availability Check"* the service should be roll-backed. Service composition is the most fault-prone activity in SBS since the task of the composition integrates divergent web services discovered through different descriptors. It is functionally very expensive and not significant towards end level solutions, and may lead to serious vulnerable [46]. Migrating the failed execution into a best alternative execution of the composite service which has the same ability to reach a final state. Computing the number of invisibly compensated transitions is NP-complete. Runtime unavailability of component services may result in composition failure. Menadjelia [38] has developed protocol-based automatic failure recovery to recover composite service unavailability problem. The finite state machine has been used to model the approach.

5.2.4. Binding Fault

When service provider tries to bind the agreement service in the system of service consumer from the service repository, there may happen various kinds of faults like mismatching faults, authentication, authorization and accounting faults. Signature mismatch faults, SLA faults, security-related faults are identified as binding faults. Mainly, there are two types of faults considered as binding faults. They are *signature mismatch* and *SLA fault*.

If the signature assigned to service consumer does not match the signature of service repository, then there will be *a signature mismatch* problem. *SLA fault* is associated with the problem while making an agreement between service broker or provider to the service consumer. Kandukuri *et al.* [31] have stated that customers participating in malicious or aggressive internet activities do not guarantee for SLA claims and shall be in violation of the AUP (Acceptable Use Policy). False or repetitive claims are also a violation of the terms of service and may be subject to service suspension. Whenever an SLA violation occurs to a service it can impact dependent services. Ismail *et al.* [33], [34] have proposed SLA violation handling approach using incremental time impact analysis. Their approach can efficiently recover and handle the violation in relation to the strategy of minimizing the number of service changes. It finds impact region, existing impact region and expands the impact region, and increases the range if it does not cover the affected range.

5.2.5. Execution Fault

Failures of system on-time completion are caused by uncertain system performance. It is very difficult to find the location of the temporal violation where exactly in a service. Luo *et al.* [37] have proposed temporal violation handling point selection strategy to deal with the temporal violation. They have adopted throughput consistency state verification and violation handling point selection. Transactions have been evolving from flat transactions, transactions with save-points, nested transactions, and more advanced ones, in order to give it more flexibility. Some transactions may be long-lived transactions that exploit the resources and can hold locks on external objects for long periods of time due to the infinite execution in the faulty situation. So, Balbastro *et al.* [19] have discussed transaction isolation fault and proposed a mechanism for handling exceptions in a synchronized way.

5.2.6. Service Delivery Fault

We have categorized service delivery fault as separate fault category because it is also an emphasized stage of the SBS. Many researchers have primarily been focused on service delivery fault. Balbastrol *et al.* [19] have developed CAA-DRIP framework as fault tolerant system to deal with latent errors and dormant faults and errors at service delivery in SOA. They have used simulation technique as repair assumption. Timeout error is also very common and occurs in any distributed system. Yeung *et al.* [29] have proposed the formal approach to handle timeout exception and message events. If an invocation cannot complete within the maximum duration allowed, the event handler will take over and halt the process.

5.3. Non-functional Faults

At runtime, some services may become faulty and cause a process to disrupt its end-to-end quality of service (QoS) constraints. The faults identified in this category are also common with distributed system faults and SOA cycle-specific faults, however, we have specified only non-functional and security-related faults in this category. If the system ensures the security, then the SBS can be more dependable and reliable, which guarantees the overall quality of service. Thus, we have categorized security related faults into this class. Security fault, SLA (Service Level Agreement) and QoS are briefly explained as follows.

5.3.1. Security Fault

Security in SBS is important so as to ensure reliable operation and to protect the integrity of stored information. In case of authentication, authorization and accounting problem, security fault may occur. Violation of security requirements includes integrity, authentication, non-repudiation and confidential violations [3]. Confidential violations cause invalid service invocation and intruder may attack the system. Authentication ensures whether the service user is authenticated to bind the service. If the authentication detail provided by the user is not valid, then there will be authentication fault. For security purpose, sensitive data should be in encrypted form, but there may be encryption fault and/or decryption fault to the service due to invalid request or design time fault. If the signature provided by the service user is invalid, then there will be a signature fault.

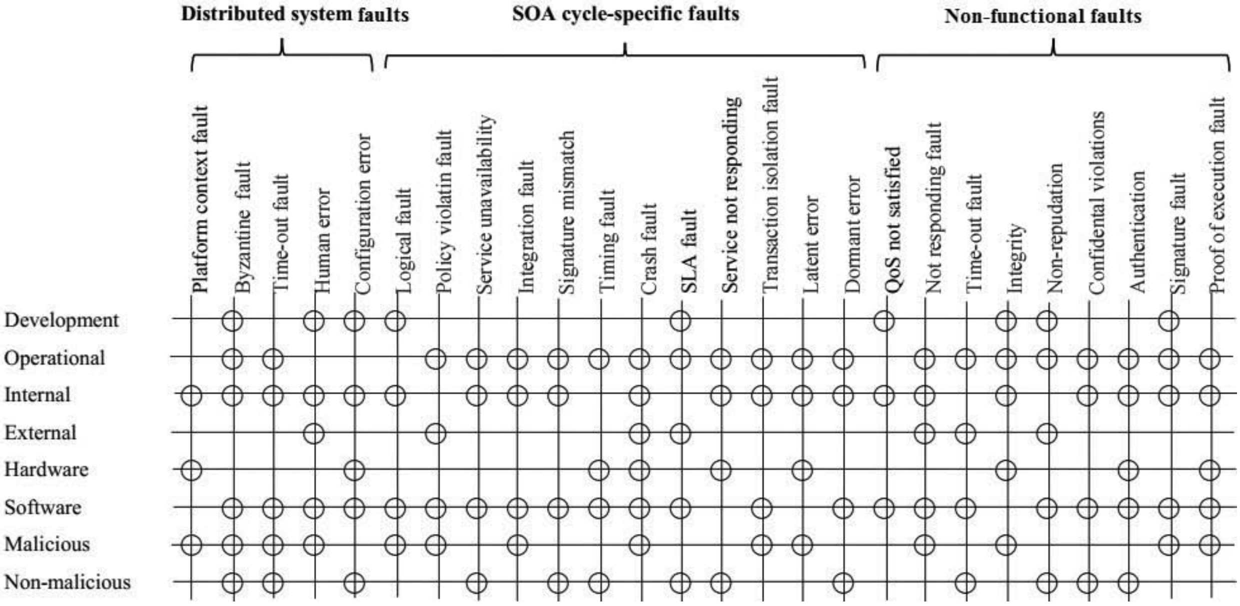
5.3.2. SLA Fault

SLA (Service Level Agreement) is a commitment between the service provider and the service consumer confirming the minimum levels of service to be expected from a particular product. Whenever SLA violation occurs to a service it can impact dependent services. SLA fault is also noticed in binding because, while the service consumer is able to use the service as defined in SLA, there can be SLA fault. SLA claim fault, as defined by Kandukuri *et al.* [31], false and repetitive claims by service customers are also a violation of the terms of service and may be subject to service suspension. A customer participating in malicious or aggressive internet activities does not ensure for SLA claims and shall be in violation of the AUP (Acceptable Use Policy). There are some studies conducted on handling SLA violation in SBS as in [33].

5.3.3. QoS Not Satisfied

QoS (Quality of Service) is the measure of transmission quality and service availability of a network [47]. Service availability is a major foundation element of QoS. The design of high availability of the network infrastructure ensures the QoS. Loss, delay, jitter, low bandwidth etc. are the major parameters of network traffic that can increase the downtime of the SBS. Delay Variation (Jitter) between the end-to-end services due to the unstable communication can generate the time-out fault [47]. A system is said to be reliable if every packet in the system experienced the bounded delay [48]. Due to a low bandwidth, message passing can take undesirably long time. QoS *not satisfied* is directly integrated with other faults.

In a network of services, failure of one service can cause the other dependent services failure. Lhaksmana *et al.* [36] have addressed cascading failure in the service network and provided cascading failure tolerance approach. By means of the cascading failure simulation they have found that scale-free topology shows better tolerance, the effect of network topology on tolerance is more significant at a lower degree of alternative services. The inverse of the degree of alternative is increased as the number of nodes experiencing cascading failure increases. The number of nodes involved in cascading failure is estimated linear to the average number of required component services. Luo *et al.* [37] have proposed a temporal violation handling from throughput consistency state verification and selection of the violation handling point which is able to determine the location of temporal violation in service and workflow.



	Distributed system faults					SOA cycle-specific faults												Non-functional faults								
	Platform context fault	Byzantine fault	Time-out fault	Human error	Configuration error	Logical fault	Policy violatin fault	Service unavailability	Integration fault	Signature mismatch	Timing fault	Crash fault	SLA fault	Service not responding	Transaction isolation fault	Latent error	Dormant error	QoS not satisfied	Not responding fault	Time-out fault	Integrity	Non-repudation	Confidental violations	Authentication	Signature fault	Proof of execution fault
Development		⊕		⊕	⊕	⊕							⊕					⊕			⊕	⊕			⊕	
Operational		⊕	⊕				⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕		⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Internal	⊕	⊕	⊕	⊕	⊕	⊕		⊕	⊕	⊕		⊕		⊕	⊕	⊕	⊕	⊕	⊕		⊕		⊕	⊕	⊕	⊕
External				⊕			⊕					⊕	⊕						⊕	⊕		⊕				
Hardware	⊕				⊕						⊕	⊕		⊕		⊕					⊕			⊕		⊕
Software		⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕		⊕		⊕	⊕	⊕	⊕		⊕	⊕	⊕	⊕	⊕
Malicious	⊕	⊕	⊕	⊕		⊕	⊕		⊕			⊕			⊕	⊕			⊕		⊕				⊕	⊕
Non-malicious		⊕	⊕		⊕			⊕		⊕	⊕		⊕	⊕			⊕			⊕		⊕	⊕	⊕		

Figure 5. Matrix representation of the proposed extended fault taxonomy.

Matrix representation of proposed extended fault taxonomy is presented in Figure 5. It shows the interaction between SOA faults and dimensions where dimensions are adopted from Avizienis *et al.* [6]. There are eight dimensions included in our proposed fault taxonomy: development and operational, internal and external, hardware and software, and malicious and non-malicious, as shown in rows. Θ symbol indicates that there is an interaction between dimensions and the corresponding fault in the column. If there is no Θ symbol, then there is no interaction between dimensions and the fault. One fault can occur in several dimensions at a time. For example, crash fault can be the operational fault, and/or internal fault and/or hardware fault and/or malicious fault. A fault taxonomy proposed in [9] also tried this in same way adopting dimensions from Avizienis [6] but their taxonomy has some limitations on overlapping. In their work, there is no overlapping on the dimensions of the same group like time-out fault can be hardware fault and software fault as well but their taxonomy is unaware of this.

6. Fault Recovery Strategies for SOA-based System

We have categorized fault recovery strategies of service-oriented computing into two major categories: local recovery strategies and global recovery strategies, as shown in Figure 6. Internal recovery concerns the interactions among parameters in a service. Forward recovery technique is related to the transactional behaviors of the messages as results all or nothing. Backward recovery is associated with faults occurring in situation where multiple services interact with each other. It applies any of the exception handling strategies like ignore, wait, retry, recompose, retryUntil etc. Backward recovery means rollback of the faulty service with the previous healthy version of the same service. Forward recovery is more optimized and has better performance than backward recovery in service recomposition and recreation. Forward recovery technique either ignores the fault service and goes forward to keep the rest of the system running with no harm or retries the faulty service again or substitutes the faulty service with the another service which would be sufficient to fulfill the task of the current faulty service. Four types of replication/repetition mechanisms [49] can be used for fault tolerance system: passive repetition, active repetition, N-version model, and return to back/check-point model. Brief explanations of local recovery strategies and global recovery strategies are as follows.

6.1. Local Recovery Strategies

Local recovery strategies try to fix the fault in the current state of error. After successful correction, the system tries to continue its normal execution from the same state. Ignore, notify, halt, terminate and redundancy are noticed as local recovery strategies which are also shown in Figure 6.

- *Ignore*: Ignore strategy just ignores the identified faults that do not affect the whole system, and does not violate the goal. It is an effective action in case of performance utilization and reliable system if the fault is temporal.
- *Replace*: In case of service fault, replace action replaces the faulty service by an alternative equivalent service with the same functioning. The replace action might call for compensation or rollback to recover.
- *Retry*: It retries the fault generating service repeatedly, till the maximum retry times have been exhausted. Web server is stateless between transactions; it does not maintain important state from first and last. The requests being processed are effectively dropped. A client may or may not receive a response that completely relies upon the in-process requests. The re-issuing of the request can lead to further problems since the same request may then be executed multiple times.
- *Reboot*: Shorting down the system and re-executing from the beginning to reduce the unstable state problem in the same environment increases the cost. It is a time-consuming process to reboot the whole system so Candea *et al.* [50] have proposed Microreboot as a technique to increase the overall performance of the fault recovery rather than whole system reboot. Rather than rebooting the whole system Microreboot just reboots the particular module.

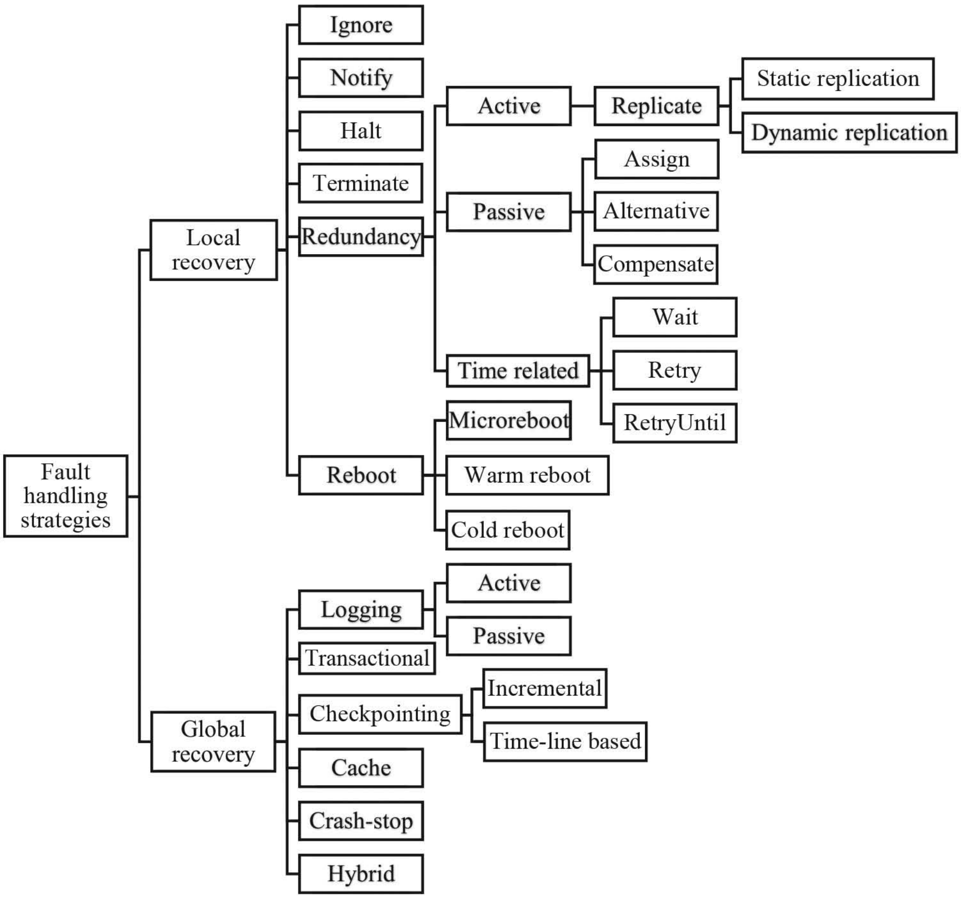


Figure 6. An effective summarization of fault recovery strategies.

- *Recompose*: This action searches for the alternative process with the same objective, discarding the current faulty process. It may be the last option while repairing the faulty service because it is the most time-consuming fault handling strategy. But this strategy is suitable for all fault handling recovery cases.

- *Replication*: Replicating the same service or process in several systems as a backup is a common approach to fault recovery. Fault service can be replaced by the same version of service from the backup system. N-version programming can be used to implement replication. As identified by Mohamed [51], there are two types of replication: *static replication* and *dynamic replication*. In *static replication*, the number and position of the replica are fixed over time and do not change their behavior during runtime. When a single replica becomes unresponsive, this replica is still considered a member of the replica communication group. *Dynamic replication* adapts to a dynamic flexibility with the number of online replicas, their physical locations and selection of active replica during runtime [51]. Replication technique, as a pre-emptive strategy to reduce the SBS fault, has some drawbacks. It increases the time complexity and space complexity to maintain the replicas for every request of the client. Also it needs request synchronization between servers to be deterministic.

6.2. Global Recovery Strategies

Logging, transactional, checkpointing, cache, crash-stop, hybrid strategies are considered as global recovery strategies. Figure 6 shows some of the popular global recovery strategies. These are briefly explained in the following subsections.

- *Logging*: Logging mechanism stores intercepted message traces of every transaction of service interactions. Later, if a fault occurs, message traces can be used for fault repairing purpose. Its main concept is to redundantly store or log all the messages delivered to the primary server on stable storage or a replica.
- *Transactional*: Protocol design to do with service atomicity using transaction integrity concept is used in this strategy. In case of a fault, the current process returns to the stable state before executing the interaction protocol which is known as skip processing strategy to accomplish the fault recovery.
- *Checkpointing*: In the checkpointing approach, the server state is periodically copied, either to a standby server(s) or to a stable storage. There are basically two checkpointing approaches as mentioned by Ayari *et al.* [52] – incremental checkpointing and time-line checkpointing. Incremental checkpointing tries to maximizing the consistency of the replicated states by performing checkpoints each time a critical state change occurs at the primary code. This approach increases the complexity. Another approach is time-line based checkpointing where a state is checkpointed each period of time. Time-to-checkpoint value depends on the measured failure frequency.
- *Cache*: Possible state inconsistencies, compensated by state-caching and retrying only failed interactions ensures the lower performance overhead on the scalable infrastructure. Wang *et al.* [24] firstly proposed cache-based process transformation using Petri nets to find the circular dependency. This strategy caches only the response message to achieve robust client/server interaction, unlike backup or *n*-version programming.
- *Crash-stop*: SOA-based computing systems may fail permanently in an interrelated fashion at any random instant following the so-called crash-stop failure model where tasks cannot be recovered from a failed server. If we leave the system uncontrolled, then the bad situation may be worse and the amount of cost increases, so in this condition, it is better to stop the current functioning of the system.
- *Hybrid strategies*: In order to establish more secure recovery, computing two or more strategies can be combined to recover faulty situation in SBSs. Many researchers have practiced hybrid strategies rather than optimizing a particular strategy. A hybrid technique with application-level logging and connection replication, named CORAL (A Client-Transparent Fault-tolerant) mechanism is proposed in [53]. CORAL recovers in-process requests and does not require deterministic servers, or changes to the clients. To achieve the fault tolerance goals, active replication of the servers may be used, where every client request is processed by two (or more) server replicas. Logging of the request is an alternative, but two different replies for the same request may reach the client violating the requirement for transparent fault tolerance. Their approach has assumed that only one host at a time can be affected by the fault and the impact of the fault can be to either crash a process or crash or hang the entire host. Rollback and compensation are analogous to their usual definitions.

7. Challenges on SBS Fault Handling

In this section, we highlight some challenges observed in the literature. Some major challenges like security and interoperability have gained a lot of attention by the researchers, whereas others like interoperability, availability, performance and scalability also require more attention. Some fault handling approaches have been tested in laboratories, some are just simulated, but when they come to real applications, any of the new challenges may arise. Wang *et al.* [24] have mentioned consistency and robustness of the service as a challenge. Ismail *et al.* [27] have mentioned maintainability as a challenge in fault handling if some services have to be suspended for a few reasons such as maintenance purposes, unlike 24/7 availability of services. Some might be available only

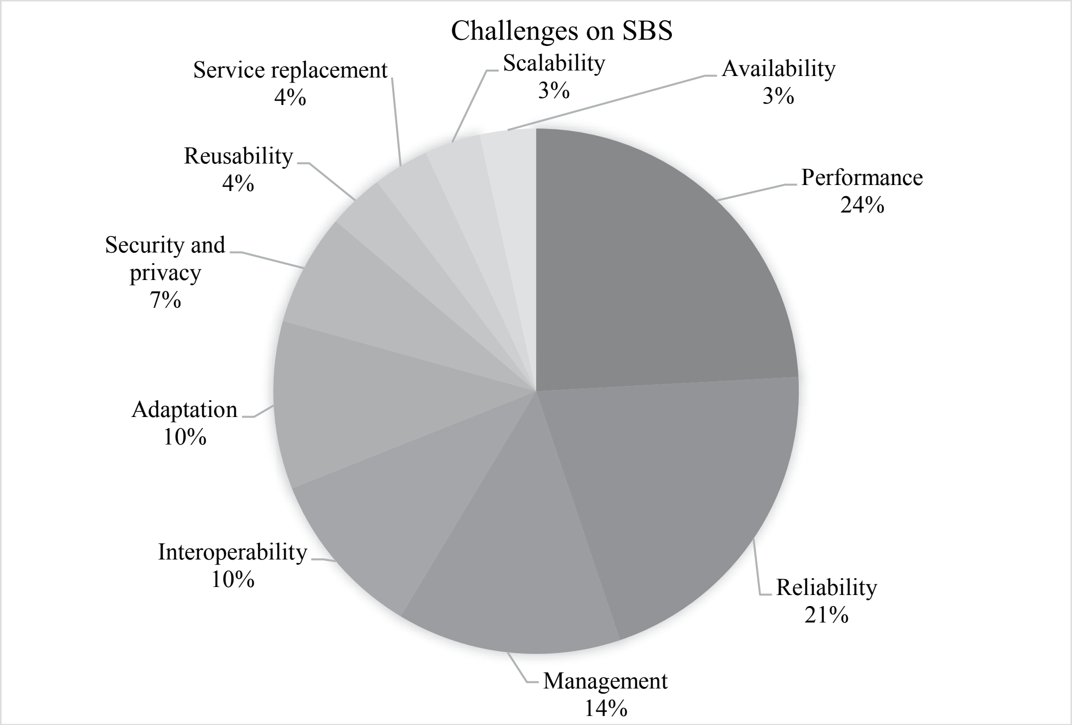


Figure 7. Challenges on SBS fault handling.

during certain operating hours. In our previous work [54], from the literature review on the basis of distributions of papers in fault handling of SBS, we observed performance and reliability as major challenges in fault handling which is as shown in Figure 7.

Dynamicity and adaptiveness limit the ability of the tester to determine the WSs, that are invoked during the execution of a workflow [28], [35], [55]. The adaptable system can adjust to other environments at the real time. The service-oriented system is also adaptable system. Thus, it is expensive, time-consuming and resource-consuming to detect and correct fault in different environments.

Interoperability is the key challenge [56], [57]. In SBS, interoperability always gets high concern as a challenge. Interoperability becomes a challenge as it needs to handle a large number of heterogeneous services that may belong to different, diversified platforms. Some research models or projects have been found in the literature review like WSRel, WSCol [58]. They have explored the challenges and drawbacks of the SOA and provided the appropriate strategies for fault handling. White-box testing is also a challenge for SBS fault handling, due to its inclusion of specific instructions, concurrency, fault compensation and dynamic service discovery and invocation [59]. Service developers and service providers are responsible stakeholders to consider this interoperability issue to ensure the proper delivery of services to all service customers, no matter on what the hardware/software specifications are. The following paragraphs give a brief discussion about key challenges on fault handling of SBSs facedin detection and development of faults of SOA and identify reasons and models/projects proposed by the researchers.

In general, *reliability* is the probability that a system functions correctly for a given time period. Overall system reliability increases with service redundancy that increases the rate of service delivery to the customer. Reliability of services is related to security and service availability. Infrastructures like software, hardware and network channel should be trustworthy to enrich reliability throughout all the layers of SOA. Unreliable communication may lead to system failures, data loss, and long delays. Several researchers in the studies [60], [61], [62], [63] have proposed reliability checking models to achieve service compositions. A paper [60] has concentrated on reliability issue of SBS and proposed a framework and prototype tool for detecting anomalous services in Open Service Gateway Initiation(OSGi)-based applications. Anomalous services decrease the reliability. Evaluating reliability is not an easy task because different vendors are usually black-box components which lack source code and design documents which makes it difficult to evaluate their quality by static code analysis.

Scalability is another challenge in handling faults for SBS. The scalability of the SBS means the ability to add new services, components and functions for service consumers, without negatively affecting the quality of existing services. Increasing the functionality is always a difficult task in case of heterogeneous SBS platforms and communication protocols, for example, vertical service delivery. SBSs must provide scalable mechanisms for registration, discovery of service faults as well as service interoperability.

Availability of service means the ability of SBSs to provide services to service customers anywhere and anytime. Several solutions [62], [64], [51] to achieve high availability of SOA services are to providing redundancy for services, logging, and replacement. Service availability means availability of service for service customer all the time. Some tools [63], [65], [66] can help system to maximize the service availability. Infrastructure availability means 24/7 hardware and software availability of SBS.

Due to *heterogeneous systems* [67], it is not easy to ensure security and privacy of users. So security is a significant challenge [25] for the SBS fault handling. Lack of common standard and architecture creates a problem in providing security. One approach could be access control on the application layer of SOA.

Mobility is another challenge for SBS realization because most of the services are expected to be delivered to mobile users. Mobile users [67] can move from one place to another, which may lead to temporary unavailability of service due to the devices transfer from a gateway to another gateway. For example, Internet of Vehicles [67], Ad-hoc etc.

Observability analyses the SBS dynamics and determines if the technology under consideration requires extension or enhancements to build scalable resource management solution [68]. Observability addresses how fine-grained the state of a system and its components can be observed from the outside.

Autonomy: services are autonomous means services exercise a high level of control over their underlying runtime execution environment. Service autonomy increases a service's runtime reliability, performance and predictability, especially when being reused and composed [69]. A high level of control over how service logic is designed and developed at implementation level is required.

8. Conclusion

Since SOA (Service Oriented Architecture) is a black-box in nature, the fault can be detected only when it really executes. We proposed an extended fault taxonomy that systematically presents a brief description of possible faults in SBS. The taxonomy also shows the interaction among different faults, how a fault can cause other ones. It divides the faults into three categories: SOA life cycle-specific faults, distributed system faults and non-functional faults. The knowledge of fault taxonomy of SBS and its associated challenges is essential for developing and testing fault-tolerant and dependable systems. Practitioners and researchers can obtain a general understanding of SBS dependability. To enhance the dependability of SBS, fault recovery techniques should also be considered inherently. Possible fault recovery strategies are also presented in the paper. For further work, we have a plan to extend our fault taxonomy to include possible faults in cloud computing and Internet of things.

Acknowledgment

The authors would like to thank ICCR, Ministry of Foreign Affairs, India (Silver Jubilee Scholarship Scheme) for providing funds and DST-CIMS, Institute of Science, BHU, India for providing necessary infrastructure and facilities for undertaking this research work.

References

[1] H. J. La and S. D. Kim, "Static and Dynamic Adaptations for Service-based Systems", *Inf. Softw. Technol.*, vol. 53, no. 12, pp. 1275–1296, 2010. https://doi.org/10.1016/j.infsof.2011.06.001

[2] Ieee, "IEEE Standard Glossary of Software Engineering Terminology", *Office*, vol. 121990, no. 1, pp. 1, 1990. https://doi.org/10.1109/IEEESTD.1990.101064

[3] M. X. Wang *et al.*, "Integrated Constraint Violation Handling for Dynamic Service Composition", *SCC 2009 – 2009 IEEE Int. Conf. Serv. Comput.*, pp. 168–175, 2009. https://doi.org/10.1109/SCC.2009.31

[4] S. Brüning *et al.*, "A Fault Taxonomy for Service-oriented Architecture", *Proc. IEEE Int. Symp. High Assur. Syst. Eng.*, pp. 367–368, 2007. https://doi.org/10.1109/HASE.2007.46

[5] D. W. Cheun *et al.*, "A Taxonomic Framework for Autonomous Service Management in Service-Oriented Architecture", *J. Zhejiang Univ. Sci. C*, vol. 13, no. 5, pp. 339–354, 2012. https://doi.org/10.1631/jzus.C1100359

[6] A. Avižienis *et al.*, "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004. https://doi.org/10.1109/TDSC.2004.2

[7] L. Mariani, "A Fault Taxonomy for Component-based Software", *Electron. Notes Theor. Comput. Sci.*, vol. 82, no. 6, pp. 61–71, 2003. https://doi.org/10.1016/S1571-0661(04)81025-9

[8] W. Hummer *et al.*, "Deriving a Unified Fault Taxonomy for Event-based Systems", in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems – DEBS '12*, 2012, pp. 167–178.

[9] K. S. M. Chan *et al.*, "A Fault Taxonomy for Web Service Composition", in *International Conference on Service-Oriented Computing ICSOC 2007. Lecture Notes in Computer Science*, 2007, pp. 363–375.

[10] A. Avižienis *et al.*, "Basic Concepts and Taxonomy of Dependable and Secure Computing", *IEEE Trans. Dependable Secur. Comput.*, vol. 1, no. 1, pp. 11–33, 2004. https://doi.org/10.1109/TDSC.2004.2

[11] T. Aslam *et al.*, "Use of A Taxonomy of Security Faults", *Proc. of the 19th Natl. Inf. Syst. Secur. Conf.*, pp. 551–560, 1996.

[12] G. Vijayaraghavan and C. Kaner, "Bug Taxonomies: Use them to Generate Better Tests", *Softw. Test. Anal. Rev.*, pp. 1–40, 2003.

[13] B. Kidwell and J. Hayes, "Toward a Learned Project-specific Fault Taxonomy: Application of Software Analytics", *2015 IEEE 1st Int. Work. Softw. Anal.*, pp. 1–4, 2015. https://doi.org/10.1109/SWAN.2015.7070479

[14] IBM, "Service Oriented Architecture(SOA): Simply good design", *Ibm*, 2016.

[15] Microsoft, "Chapter 1: Service Oriented Architecture (SOA)", *Microsoft*, 2016. [Online]. Available: http://www.opengroup.org/soa/source-book/soa/soa.htm

[16] Z. Huang *et al.*, "Performance Diagnosis for SOA on Hybrid Cloud using the Markov Network Model", *Proc. – IEEE 6th Int. Conf. Serv. Comput. Appl. SOCA 2013*, pp. 17–24, 2013. https://doi.org/10.1109/SOCA.2013.55

[17] W. Zhao, "Design and Implementation of a Byzantine Fault Tolerance Framework for Web Services", *J. Syst. Softw.*, vol. 82, no. 6, pp. 1004–1015, 2009. https://doi.org/10.1016/j.jss.2008.12.037

[18] F. Belli and M. Linschulte, "Event-driven Modeling and Testing of Real-time Web Services", *Serv. Oriented Comput. Appl.*, vol. 4, no. 1, pp. 3–15, 2010. https://doi.org/10.1007/s11761-010-0056-5

[19] F. Balbastro *et al.*, "Analysis and Framework-based Design of a Fault-tolerant Web Information System for M-health", *Serv. Oriented Comput. Appl.*, vol. 2, no. 2–3, pp. 111–144, 2008. https://doi.org/10.1007/s11761-008-0026-3

[20] K. Zhai *et al.*, "Prioritizing Test Cases for Regression Testing of Location-based Services: Metrics, Techniques, and Case Study", *IEEE Trans. Serv. Comput.*, vol. 7, no. 1, pp. 54–67, 2014. https://doi.org/10.1109/TSC.2012.40

[21] G. Friedrich *et al.*, "Exception Handling for Repair in Service-based Processes", *IEEE Trans. Softw. Eng.*, vol. 36, no. 2, pp. 198–215, 2010. https://doi.org/10.1109/TSE.2010.8

[22] J. Chen *et al.*, "A Web Services Vulnerability Testing Approach Based on Combinatorial Mutation and SOAP Message Mutation", *Serv. Oriented Comput. Appl.*, vol. 8, no. 1, pp. 1–13, 2014. https://doi.org/10.1007/s11761-013-0139-1

[23] C. Ye and H. A. Jacobsen, "Whitening SOA Testing via Event Exposure", *IEEE Trans. Softw. Eng.*, vol. 39, no. 10, pp. 1444–1465, 2013. https://doi.org/10.1109/TSE.2013.20

[24] L. Wang *et al.*, "Robust Client/Server Shared State Interactions of Collaborative Process with System Crash and Network Failures", *Proc. – IEEE 10th Int. Conf. Serv. Comput. SCC 2013*, pp. 192–199, 2013. https://doi.org/10.1109/SCC.2013.39

[25] W. She *et al.*, "Security-aware Service Composition with Fine-grained Information Flow Control", *IEEE Trans. Serv. Comput.*, vol. 6, no. 3, pp. 330–343, 2013.
https://doi.org/10.1109/TSC.2012.3

[26] M. Bartoletti *et al.*, "Semantics-based Design for Secure Web Services", *IEEE Trans. Softw. Eng.*, vol. 34, no. 1, pp. 33–49, 2008.
https://doi.org/10.1109/TSE.2007.70740

[27] A. Ismail *et al.*, "Analyzing Fault-impact Region of Composite Service for Supporting Fault Handling Process", *Proc. – 2011 IEEE Int. Conf. Serv. Comput. SCC 2011*, pp. 290–297, 2011.
https://doi.org/10.1109/SCC.2011.51

[28] V. Garousi *et al.*, "Traffic-aware Stress Testing of Distributed Real-time Systems based on UML Models using Genetic Algorithms", *J. Syst. Softw.*, vol. 81, no. 2, pp. 161–185, 2008.
https://doi.org/10.1016/j.jss.2007.05.037

[29] W. L. Yeung, "Formalizing Exception Handling in WS-CDL and WS-BPEL for Conformance Verification", *IEEE Int. Conf. Serv. Comput. Appl. SOCA '09*, vol. 0, no. c, pp. 262–269, 2009.
https://doi.org/10.1109/SOCA.2009.5410265

[30] A. Liu *et al.*, "FACTS: A Framework for Fault-tolerant Composition of Transactional Web Services", *IEEE Trans. Serv. Comput.*, vol. 3, no. 1, pp. 46–59, 2010.
https://doi.org/10.1109/TSC.2009.28

[31] B. R. Kandukuri *et al.*, "Cloud Security Issues", *Proc. 2009 IEEE Int. Conf. Serv. Comput.*, pp. 517–520, 2009.
https://doi.org/10.1109/SCC.2009.84

[32] M. Sama *et al.*, "Multi-layer Faults in the Architectures of Mobile, Context-aware Adaptive Applications", *J. Syst. Softw.*, vol. 83, no. 6, pp. 906–914, 2010.
https://doi.org/10.1016/j.jss.2009.11.005

[33] A. Ismail *et al.*, "Incremental Service Level Agreements Violation Handling with Time Impact Analysis", *J. Syst. Softw.*, vol. 86, no. 6, pp. 1530–1544, 2013.
https://doi.org/10.1016/j.jss.2013.01.052

[34] R. Alsoghayer and K. Djemame, "Resource Failures Risk Assessment Modelling in Distributed Environments", *J. Syst. Softw.*, vol. 88, no. 1, pp. 42–53, 2014.
https://doi.org/10.1016/j.jss.2013.09.017

[35] H. Chai and W. Zhao, "Byzantine Fault Tolerance for Services with Commutative Operations", *Proc. – 2014 IEEE Int. Conf. Serv. Comput. SCC 2014*, pp. 219–226, 2014.
https://doi.org/10.1109/SCC.2014.37

[36] K. M. Lhaksmana *et al.*, "Cascading Failure Tolerance in Large-Scale Service Networks", *Proc. – 2015 IEEE Int. Conf. Serv. Comput. SCC 2015*, pp. 1–8, 2015.
https://doi.org/10.1109/SCC.2015.11

[37] H. Luo *et al.*, "Where to Fix Temporal Violations: A Novel Handling Point Selection Strategy for Business Cloud Workflows", *Proc. – 2016 IEEE Int. Conf. Serv. Comput. SCC 2016*, pp. 155–162, 2016.
https://doi.org/10.1109/SCC.2016.27

[38] N. Menadjelia, "Towards a Formal Study of Automatic Failure Recovery in Protocol-based Web Service Composition", *Serv. Oriented Comput. Appl.*, vol. 10, no. 2, pp. 173–184, 2016.
https://doi.org/10.1007/s11761-015-0176-z

[39] F. Montagut and R. Molva, "Bridging Security and Fault Management within Distributed Workflow Management Systems", *IEEE Trans. Serv. Comput.*, vol. 1, no. 1, pp. 33–48, 2008.
https://doi.org/10.1109/TSC.2008.3

[40] G. Kola *et al.*, "Faults in Large Distributed Systems and What We Can Do About Them", *Euro-Par 2005 Parallel Process*, 2005.

[41] S. Avedaño, "Safety and Dependability Analysis to Complement Testing of Safety-critical Software", *Softcare*, 2004.

[42] H. Chai *et al.*, "Toward Trustworthy Coordination of Web Services Business Activities", *IEEE Trans. Serv. Comput.*, vol. 6, no. 2, pp. 276–288, 2013.
https://doi.org/10.1109/TSC.2011.57

[43] L. Shwartz *et al.*, "Quality of IT Service Delivery #x2014 – Analysis and Framework for Human Error Prevention", *Serv. Comput. Appl. (SOCA), 2010 IEEE Int. Conf.,* pp. 1–8, 2010.
https://doi.org/10.1109/SOCA.2010.5707161

[44] D. Nadkarni *et al.*, "Failure Analysis for Composition of Web Services Represented as Labeled Transition Systems", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2011, vol. 6551 LNCS, pp. 161–175.

[45] O. Bushehrian *et al.*, "A Workflow-Based Failure Recovery in Web Services Composition", *J. Softw. Eng. Appl.*, vol. 5, no. 2, pp. 89–95, 2012.
https://doi.org/10.4236/jsea.2012.52014

[46] S. Varadi and G. A. Rao, "Quality of Service Centric Web Service Composition: Assessing Composition Impact Scale towards Fault Proneness", *Glob. J. Comput. Sci. Technol. C Softw. Data Eng.,* vol. 14, no. 9, 2014.

[47] T. Szigeti *et al.*, "End-to-End QoS Network Design: Quality of Service for Rich-Media & Cloud Networks", Second Edition, Video Enhanced Edition, Cisco Press, 2013.

[48] P. K. Mishra *et al.*, "QoS Analysis in Data Network : Stability , Reliability , QoS Invoke Rate Perspectives", in *ICEIT Conference on Advances in Mobile Communications, Networking and Computing February*, 2017, pp. 107–111.

[49] F. Mahdian and V. Rafe, "Different Models of Dependable Services in Service-oriented Architecture", in *ICACTE 2010 – 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, 2010, vol. 1, pp. 217–220.

[50] E. A. Brewer, "Lessons from Giant-scale Services", in *IEEE Internet Computing*, 2001, vol. 5, no. 4, pp. 46–55.

[51] M. F. Mohamed, "Service Replication Taxonomy in Distributed Environments", *Serv. Oriented Comput. Appl.*, vol. 10, no. 3, pp. 317–336, 2016. https://doi.org/10.1007/s11761-015-0189-7

[52] N. Ayari *et al.*, "Fault Tolerance for Highly Available Internet Services: Concepts, Approaches, and Issues", *IEEE Commun. Surv. Tutorials*, vol. 10, no. 2, pp. 34–46, 2008. https://doi.org/10.1109/COMST.2008.4564478

[53] N. Aghdaie and Y. Tamir, "CoRAL: A Transparent Fault-tolerant Web Service", *J. Syst. Softw.*, vol. 82, no. 1, pp. 131–143, 2009. https://doi.org/10.1016/j.jss.2008.06.036

[54] G. P. Bhandari and R. Gupta, "Fault Repairing Strategy Selector for Service-Oriented Architecture", *I.J. Mod. Educ. Comput. Sci. Mod. Educ. Comput. Sci.*, vol. 6, no. 6, pp. 32–39, 2017. https://doi.org/10.5815/ijmecs.2017.06.05

[55] A. D. Brucker and J. Julliand, "Editorial: Editorial for the Special Issue of STVR on Tests and Proofs Volume 2: Tests and Proofs for Improving the Generation Time and Quality of Test Data Suites", *Softw. Test. Verif. Reliab.*, vol. 24, no. 8, pp. 591–592, 2014. https://doi.org/10.1002/stvr.1558

[56] M. Jensen, "A Fault Propagation Approach for Highly Distributed Service Compositions", *Proc. – 2008 IEEE Int. Conf. Serv. Comput. SCC 2008*, vol. 2, pp. 507–510, 2008. https://doi.org/10.1109/SCC.2008.38

[57] A. Benharref *et al.*, "Efficient Traces' Collection Mechanisms for Passive Testing of Web Services", *Inf. Softw. Technol.*, vol. 51, no. 2, pp. 362–374, 2009. https://doi.org/10.1016/j.infsof.2008.04.007

[58] L. Baresi and S. Guinea, "Self-supervising BPEL Processes", *IEEE Trans. Softw. Eng.*, vol. 37, no. 2, pp. 247–263, 2011. https://doi.org/10.1109/TSE.2010.37

[59] Z. Zakaria *et al.*, "Unit Ttesting Approaches for BPEL: A Systematic Review", in *Proceedings – Asia-Pacific Software Engineering Conference, APSEC*, 2009, pp. 316–322.

[60] T. Wang *et al.*, "A Framework for Detecting Anomalous Services in OSGi-based Applications", *Proc. – 2012 IEEE 9th Int. Conf. Serv. Comput. SCC 2012*, pp. 250–257, 2012. https://doi.org/10.1109/SCC.2012.59

[61] M. S. Ali and S. Reiff-Marganiec, "Autonomous Failure-handling Mechanism for WF Long Running Transactions", *Proc. – 2012 IEEE 9th Int. Conf. Serv. Comput. SCC 2012*, pp. 562–569, 2012. https://doi.org/10.1109/SCC.2012.50

[62] Z. Wu and N. Chu, "Efficient Service Re-composition Using Semantic Augmentation for Fast Cloud Fault Recovery", *Proc. – IEEE 10th Int. Conf. Serv. Comput. SCC 2013*, pp. 176–183, 2013. https://doi.org/10.1109/SCC.2013.78

[63] E. Ruijters and M. Stoelinga, "Fault Tree Analysis: A Survey of the State-of-the-art in Modeling, Analysis and Tools", *Comput. Sci. Rev.*, vol. 15, pp. 29–62, 2015. https://doi.org/10.1016/j.cosrev.2015.03.001

[64] P. Marcu *et al.*, "Managing Faults in the Service Delivery Process of Service Provider Coalitions", *SCC 2009 – 2009 IEEE Int. Conf. Serv. Comput.*, pp. 65–72, 2009. https://doi.org/10.1109/SCC.2009.41

[65] N. Antunes and M. Vieira, "SOA-scanner: An Integrated Tool to Detect Vulnerabilities in Service-based Infrastructures", *Proc. – IEEE 10th Int. Conf. Serv. Comput. SCC 2013*, pp. 280–287, 2013. https://doi.org/10.1109/SCC.2013.28

[66] K. Goseva-Popstojanova and A. Perhinschi, "On the Capability of Static Code Analysis to Detect Security Vulnerabilities", *Inf. Softw. Technol.*, vol. 68, pp. 18–33, 2015. https://doi.org/10.1016/j.infsof.2015.08.002

[67] Z. Sanaei *et al.*, "Heterogeneity in Mobile Cloud Computing: Taxonomy and Open Challenges", *IEEE Commun. Surv. Tutorials*, vol. 16, no. 1, pp. 369–392, 2014. https://doi.org/10.1109/SURV.2013.050113.00090

[68] Ulrike Steffens, Ed., *MDD, SOA and IT-Management*, GITO mbH Verlag, 2009.

[69] G. Lavanchy *et al.*, "Habitat Heterogeneity Favors Asexual Reproduction in Natural Populations of Grassthrips", vol. 70, no. 8. 2016.

Received: March 2017
Revised: December 2017
Accepted: December 2017

Contact addresses:
Guru Prasad Bhandari
DST-CIMS, Institute of Science,
Banaras Hindu University,
Varanasi
India
guru.bhandari@gmail.com

Ratneshwer Gupta
School of Computer & Systems Sciences,
Jawaharlal Nehru University,
New Delhi
India
ratnesh@mail.jnu.ac.in

Guru Prasad Bhandari received his MCA (Master of Computer Applications) degree from the Department of Computer Science, Institute of Science, Banaras Hindu University, Varanasi, India in 2015. His research interest covers service-oriented computing, fault tolerance and reliability analysis. He is currently doing research in the area of fault analysis of service-oriented computing. He is pursuing his doctoral work under the supervision of Dr. Ratneshwer.

Dr. Ratneshwer Gupta received his PhD in Component Based Software Engineering from Indian Institute of Technology, Banaras Hindu University, Varanasi (IIT-BHU), India. His research areas are Component-Based Software Engineering and Service-Oriented Architectures. He is serving as an Assistant Professor in the School of Computer & Systems Sciences, JNU, New Delhi, India. He has been actively involved in teaching and research for the last 8 years. He has published 16 research papers in international journals and 16 research papers in international/national conference proceedings in his credit.