Survey on Web Services Fault Tolerance Approaches Based on Checkpointing Mechanisms

M. Vargas-Santiago[1*], S. E. Pomares-Hernandez[1, 2, 3], L. A. Morales Rosales[4], and H. Hadj-Kacem[5]

[1] National Institute for Astrophysics Optics and Electronic, Luis Enrique Erro # 1, Puebla, Mexico.
[2] CNRS, LAAS, 7 avenue du Colonel Roche, Toulouse, France.
[3] Univ de Toulouse, LAAS, Toulouse, France.
[4] CONACYT-Universidad Michoacana de San Nicolás de Hidalgo, Morelia, México,
[5] ReDCAD University, Sfax, Tunisia.

* Corresponding author. Tel.: +52 5516326333; email: mariano.v.santiago@ccc.inaoep.mx
Manuscript submitted April 5, 2017; accepted June 10, 2017.
doi: 10.17706/jsw.12.7.507-525

Abstract: Complex business processes are based on Web services composition (WSC). Services composition dramatically reduces the cost and risks of building new business applications. Although Web services composition has been widely researched, several issues related to dependability still need to be addressed. In this aspect, one primary concern is to provide fault handling mechanisms. Over the last decade, diverse works tackling fault tolerance for WSC have appeared; most of them are based on the checkpointing paradigm. Some of the works are oriented towards orchestration and others towards choreography. In this work, we present a study regarding the different checkpointing techniques and their applicability to WSC. This study has been done considering the different types of faults (e.g. transient, intermittent and permanent) and modes of recovery (e.g. local vs global). We introduce a novel taxonomy for fault tolerant mechanisms that groups existing works according to integration approaches, fault types and modes of recovery. We present a study of the works, illustrating their advantages and drawbacks. Finally, the paper presents a discussion and outlines several open challenges regarding fault tolerance for WSC.

Key words: Survey, web services, choreography, orchestration, fault tolerance, SOA, checkpointing, dependability.

1. Introduction

Web Services Composition (WSC) has emerged as an important computing paradigm to create complex business processes [1]. In distributed heterogeneous environments, individual Web services are used as fundamental elements to support fast and low cost development of a set of interacting services, which form comprehensive businesses functionalities [2], [3]. Thus, according to predefined business requirements, WSC refers to the process of adaptively composing a set of available Web services into a business process flow. Services composition dramatically reduces the cost and risks of building new business applications in the sense that existing business logics are represented as Web services and could be reused [4].

Web services are presented as a promising technology to implement Service Oriented-Architecture (SOA) [5]-[7]. Composing such Web services implies using standard-based languages which interact through Internet-based protocols. Notwithstanding that these technologies readily allow creating large-scale systems, they are, however, prone not only to incoming errors from the dynamic and unreliable Internet, but also to errors that increase proportionally to the number of component counts [8], [9]. Although Web

services composition has been heavily researched, several issues related to dependability still need to be addressed. In this aspect, one primary concern is to provide fault handling mechanisms [10, 11, 12, and 13]. Adopting robust fault tolerance mechanisms is necessary because they reduce the risk of faults, and businesses can properly be automated. Therefore, fault tolerant business processes are a necessity that need to be intensified because failures may lead to terrible consequences, for instance, increasing the execution time, elevating the costs of the running applications, destroying or breaching the systems [14]. Clearly, organizations need a way to guarantee consumers' needs, meaning, delivering the requested services and delivering what the services are expected to do in a timely manner. Composing Web services is achieved through integration approaches, such as choreography and orchestration or a combination of these approaches [15]-[18]. Over the last decade, diverse works tackling fault tolerance for WSC have appeared [19]. Many of them are based on the checkpointing paradigm. Nevertheless, trying to extrapolate the checkpointing paradigm into another paradigm like Web services has proven to be a complicated task due to the dynamic nature under which Web services interact, and even choosing which checkpointing technique is the most appropriate one becomes a complicated task [20], [21]. For example, in the literature we can find four different checkpointing types of mechanisms: asynchronous or uncoordinated, synchronous or coordinated, quasi-synchronous or communication-induced and message logging based checkpointing. Regardless of which checkpointing mechanism is used, rollback recovery increases the reliability and availability of distributed systems [22].

Despite different checkpointing mechanisms these have not been classified by previous literature studies [20], in this regard, fault tolerant for WSC based on checkpointing, some issues remain open:

- Let us note the lack of propositions based on checkpointing that deal with both orchestration and choreography.
- The lack of classification for the checkpointing mechanism used.
- Works based on checkpointing for new emerging trends like interactive BPEL processes and choreographies.
- A thorough work which points out advantages and drawbacks of using different types of checkpointing mechanisms.

In this paper, we present a study regarding the different checkpointing techniques and their applicability to WSC. This study has been done considering the different type of faults (e.g. transient, intermittent and permanent) and modes of recovery (e.g. local vs global). We then introduce a novel taxonomy for fault tolerance mechanisms that groups existing works according to integration approaches, fault types and modes of recovery. We present a study of the works illustrating their advantages and drawbacks. Finally, the paper presents a discussion and outlines several open challenges regarding fault tolerance for WSC.

The remainder of this paper is organized as follows. Section II presents a brief background on checkpointing mechanisms, Web services and Web services composition (orchestration and choreography), fault types and fault recovery. Section III proposes a novel taxonomy and reviews the strengths and weaknesses of the current works. Section IV gives a discussion of previous works, and some open challenges are identified. Conclusion and future work for possible trends are discussed in Section V.

2. Fundamentals and Web Service Faults Analysis

This section presents the basis for checkpointing Web services composition, giving brief definitions on concepts like: Web services characteristics, Web services composition models, checkpoint, checkpointing, rollback and their applicability to Web services composition, by emphasizing their integration and describing how these two paradigms may get along, or how one can benefit the other. Also, we present an analysis concerning the types of faults in Web services composition.

2.1. Web Services Characteristics

Standardization efforts establish the restrictions for building Web services that exhibit the following characteristics [23]:

- Web services are platform-independent and language neutral. They are accessed through a well-known interface. Therefore, Web protocols ensure effortless integration of heterogeneous distributed environments.
- Web services provide an API that can be called by other programs. This interface applies the application-to-application programming technique that can be summoned, for example, by BPEL or any other type of application. The API provides access to the application logic.
- Web services are registered through a Web service registry, which enables service consumers and organizations to easily find services that match their needs.
- Web services make interconnections flexible and adaptable because they add a layer of abstraction to the environment. Therefore, Web services support loosely-coupled connections between systems and communicate through their API by exchanging XML messages.
- Another aspect considered as non-functional requirement for Web services is:
- Quality of Service: Web service composition must agree on the level of QoS that has to be met. One open challenge is to take into consideration this nature when applying checkpointing mechanisms.

2.2. Web Services Composition Models

Service composition is fundamental in the SOA paradigm. It is oriented towards building complex Web services from smaller components. Composition rules deal with the way in which different services compose a coherent global service. In particular, they specify the order in which services are invoked and the conditions under which a certain service may or may not be invoked. The design of composing Web services is mainly carried out throughout two composition techniques, namely choreography and orchestration.

Orchestration. In this composition model, the involved Web services are under the control of a single endpoint central process (orchestrator). This process coordinates the execution of various actions on the Web services involved in the composition. The Business Process Execution Language (BPEL) [24] has become one of the predominant standards for Web services composition [25]. BPEL orchestrates the interactions between Web services within a single party that controls and describes a process flow or workflow. One core feature offered by BPEL is the support for asynchronous communications, which is needed between long-running applications based on Web services. BPEL provides an infrastructure that manages data persistence.

Three basic fault handlers are provided by the BPEL engine: compensation handlers, fault handlers, and event handlers [24].

- Compensation handlers are used to undo or reverse the effects of a previous activity, specifying the actions to be executed.
- On the other hand, fault and event handlers execute actions at runtime for predefined faults and/or events.

Nonetheless, BPEL only manages predefined faults specified by application designers.

Choreography. This composition model presents an abstract description of protocols. It offers a top view of the management rules which govern the interactions between the involved services in a decentralized application. It differs from orchestration because the former represents control from one party's perspective. It allows each involved party to describe its part in the interaction, thus, being more collaborative. Choreography tracks the message sequences among multiple parties and sources rather than

a specific business process that a single party executes [15]. It is modeled by abstract processes. An abstract process or business protocol specifies the public message exchanges between parties.

Choreography uses the Web Service Choreography Interface (WSCI) which defines a collaboration extension to the Web Services Description Language (WSDL). In other words, it defines the overall choreography or message exchange between Web services. The specification supports message correlation, sequencing rules, exception handling, transactions, and dynamic collaboration [15].

2.3. Web Services Composition Recovery Modes and Fault Types

In this section, we will first talk about the recovery modes that are common in literature for composite Web services, then we categorize fault according to behavior, instant and origin of the faults. Afterwards, we will give the recovery strategies based on checkpointing mechanisms.

Recovery modes. There are two types of recovery modes under which composite Web services are currently recovered: global recovery and local recovery mode, described as follows:

- *Global Recovery Mode*: If the overall system rolls back, not only does the failed Web service roll back,
- But so do all others which are directly or indirectly affected. This is known as global recovery (the overall system recovers). Examples of Web services composition that include this recovery mode are found in [14, 26, 27, 28, 29, and 30].
- *Local* Recovery *Mode*: This occurs when individual Web services fail and attempt to roll back to a well-known point in time where it was working properly, without needing other Web services to perform recovery actions. More efforts have been put into this kind of solution. Some examples can be found in [31, 32, 33, 34, 35, 36, 37, 38, 39, and 40].

Categories of faults. We classify the faults according to: behavioral aspects and the moment and origin of the faults. According to behavioral aspects, the faults can be grouped into permanent, intermittent, transient and byzantine faults.

Regarding byzantine faults, generally these are processes that may behave arbitrarily; these may disseminate different information to other processes, resulting or constituting a serious threat to the integrity of a system [41].

Transient faults happen once and then disappear, usually after time the system will behave normally. Intermittent faults happen, then they go away and happen again, and so on, and so forth. These faults behave sporadically and are hard to fix. Permanent faults are caused by system components and do not go away until the component is replaced. We can conclude that byzantine faults are rather cumbersome because no assumption can be made about them. These are difficult to trace and are sometimes even undetectable, for instance not knowing which server/service has failed. Regarding transient, intermittent and permanent faults, we are interested in the fail-stop faults.

In fail-stop faults, a component stops working (temporarily or permanently) but it is assumed that any correct component in the system is able to detect it. Therefore, for simplicity without a loss of generality, in the rest of the paper we classify the faults according to their behavior, into only fail-stop faults and byzantine faults. According to the moment and origin of the faults, Chan et al. in [42] have introduced a taxonomy categorizing the faults into the following classes:

- Development faults, which occur during system development or maintenance.
- Operational faults that occur during service delivery. For instance, as presented in [27, 37] where Cardinale et al consider faults during the execution process of a Transactional Composite Web Service (TCWS). However, it can be considered as a fail-stop failure since it is detectable and the authors stipulate that they use replacement of the entire failed Web service.

- Internal faults originating inside the system boundary. For example, QoS degradation faults due to a lack of resources [39]. Concerning QoS degradation detection, also grouped into fail-stop fault.
- External faults that originate outside the system boundary and are propagated into the system by interaction or interface. For instance, [14], external faults may consider integrity attacks to Web services composition.
- Hardware faults that originate in or affect the hardware. One example of this is a system crash [31]. Other examples that consider this kind of faults are presented in [34, 36].
- Software faults that affect programs or data. An example that considers this kind of faults is presented in [34].

Fault Recovery strategies. Finally, we present the most common recovery strategies found in literature overview, applied to Web services composition.

- Backward error recovery: after a failure occurs, Web services are rolled back to an existing point in time where they were functioning properly. These are commonly based on checkpointing mechanisms [14], [26], [27], [31], [33], [34], [36]-[40].
- Forward error recovery: the failed Web services are replaced using substitution and/or replacement [43], [44] of the failing or a subset of the failing Web services.

In forward error recovery, the system tries to repair the failure without stopping its execution; some techniques include retry and recovery. For example, Shuchi and Bhanodia use substitution of a subset of Web service that contains one or more failed Web services and replaces such subset with an equivalent subset [43]; however, this solution is not based on checkpointing. Only some works tackle Web services composition based on checkpointing using this kind of recovery type [27], [29]-[37]. There are two well-known techniques for fault tolerance in a distributed system: “active replication” and “passive replication” [45].

- **Active Replication:** Active Replication means creating redundant application servers. When the system receives a request from the client, the request is forwarded to all replicas, for example concerning Byzantine faults [46].
- **Passive Replication**: Passive Replication means that only one server acts as the primary one to do the assigned job. If it fails, the backup server takes over, for example [47].

In [48], Monser et al argue that active and passive replication can be done by means of checkpointing. For example, checkpointing is used by replication strategies but in different ways: passive replication uses checkpointing during normal operation. Active replication, on the other hand, do not use checkpointing during normal execution, but uses it to initialize a new recovering replica. This work describes many other technologies to increase the dependability and security of Web services. Regarding checkpointing mechanisms, they point out ways to apply it to a typical Web services architecture; however, it is a merely descriptive work and no evaluation was detailed.

2.4. Checkpointing Mechanisms and Their Applicability to Web Services Compositions

- Checkpoint: refers to the information gathered by a processor in a certain time. With such information the processor can return to that checkpoint [49].
- Consistent Global Snapshot (CGS): It identifies checkpoints that do not have a causal path; they are not related by a message or a sequence of messages.
- Rollback Recovery: it treats a distributed system application as a collection of processes that communicate over a network. It achieves fault-tolerance by periodically saving the states of a process during failure-free execution, enabling it to restart from a saved state upon a failure to reduce the amount of lost work [22].

Considering the above premises, the literature presents four different checkpointing types of mechanisms oriented towards rollback recovery, namely, asynchronous or uncoordinated checkpointing, synchronous or coordinated checkpointing, quasi-synchronous checkpointing or communication-induced and message logging. These can save checkpoints on the stable storage or on the volatile storage depending on the failure scenarios to be tolerated.

2.4.1. Asynchronous or uncoordinated checkpointing

Asynchronous checkpointing consists in each participant taking its own checkpoint, its main advantages are that it eliminates the synchronization overhead imposed by synchronizing, and it has low overhead during normal execution. The main flaw for this approach is that it is susceptible to the domino effect; it is also known in the literature as rollback propagation where processes that should not be rolled back are, in fact, rolled back.

This approach is not suitable for composite Web services because the system may revert to an inconsistent state. For instance, in a bank business process where the bank or the customer exchange money, the system suddenly fails. One possible inconsistent outcome is that a state of an account A was recorded before the transfer to account B; thus, the system may revert to a state that represents losses for the customer or the bank.

2.4.2. Synchronous or coordinated checkpointing

Synchronous checkpointing solves the domino effect flaw of uncoordinated checkpointing since a process always restarts from its most recent checkpoint; however, all processes must orchestrate their checkpoint activity to form a consistent global snapshot. The storage overhead is reduced because in such technique each process maintains only one checkpoint on the stable storage. Coordinated checkpointing guarantees checkpointing consistency in two main ways:

- *Blocking*: All processes must agree on when to take their checkpoints. An initiator sends a control message to all other processes to take their checkpoints. When receiving such message, the process can no longer send or receive messages, then takes a tentative checkpoint and acknowledges the initiator. For Web services, this is unacceptable, blocking such may result in monetary losses.
- *Non*-Blocking: Based on piggybacked information, processes decide when to take their checkpoints. For Web services composition, this technique is compatible; however, it may not be suitable because of the high overhead and the high control information used.

2.4.3. Quasi-synchronous or communication-induced checkpointing (CiC)

In quasi-synchronous checkpointing, processes take checkpoints based on the control information piggybacked on the application messages it receives from other processes. Upon the detection of dangerous patterns, like Z-paths, forced checkpoints are taken. CiC is another way to avoid the domino effect since it allows processes to take some of their checkpoints independently [22].

CiC is a well-known and studied mechanism which takes into consideration the correlation between recovery overhead in case of failures, and checkpointing. In quasi-synchronous checkpointing, processes take checkpoints based on the control information piggybacked on the application messages it receives from other processes. Upon the detection of dangerous patterns, like Z-paths, forced checkpoints are taken.

CiC is another way to avoid the domino effect since it allows processes to take some of their checkpoints independently [22]. CiC is a well-known and studied mechanism which takes into consideration the correlation between recovery overhead in case of failures, and checkpointing overhead, in case of the system failure free execution. In the case of Web services, this can be leveraged because checkpoints can be effectively generated, avoiding non-useful checkpoints.

CiC can be implemented as a transparent mechanism, meaning that it does not require modifications to target applications. In the case of composite Web services, the challenge is to leverage the CiC mechanism in

order to achieve and reduce to a minimum the causal control overhead sent per message in the communication channels. The characteristics that should be exploited by CiC for Web services composition are:

- CGSs can be formed easily as the CiC mechanism avoids dangerous patterns and guarantees consistency by means of forcing checkpoints when needed.
- The *system* will revert to the last CGS; therefore, it does not overwhelm the system with unnecessary storage.

2.4.4. Message logging based on checkpointing

Message Logging based checkpointing oriented towards rollback recovery, consists in saving or recording, by each process in a log, all received and sent messages. The main advantage of this technique is that processes that do not suffer a failure do not need to be rolled back and may continue their execution. However, recording so many messages is expensive in practice; therefore, different alternatives of this technique have been proposed. Pessimistic *Logging, Optimistic Logging and Causal Logging*. Currently, we have found in literature that most solutions for composite Web services either are based on or implement this mechanism [26], [31], [32], [50], and [51].To conclude this section, one can argue that asynchronous, synchronous and quasi-synchronous checkpointing mechanism requires all participants or processes to rollback. However, some of their advantages are that they will most likely recover from their last known CGS and guarantee a global recovery within a consistent state, except in the case of an asynchronous mechanism.

3. Fault Tolerance Techniques for Web Services Composition

With the proliferation of Web services technology within enterprises, many studies emerged for reliable service composition [44] and for composition recovery [35]; nevertheless, there is a need for a new and specific study to classify and give a taxonomy in the realm of fault tolerant Web services that apply checkpointing mechanisms. Therefore, we propose a novel taxonomy that addresses the techniques applied from the perspective of Web services composition paradigms as Fig. 1 shows. It is because Web services depend on hardware and software to function properly that the fault tolerance property must be enabled. Fault tolerance is highly desired for Web services composition because it can ensure for long running applications that they are accomplished in a timely manner.

In this section, fault tolerance approaches, drawbacks and issues for many approaches are briefly reviewed in the context of Web services composition for both integration approaches: orchestration and choreography. Fig. 1 shows an abstract view of fault tolerance techniques categorized under orchestration and choreography reviewed in terms of a new classification, namely, global and local recovery, as they are the most used fault tolerance techniques found in the literature overview.

3.1. Fault Tolerance Techniques for Orchestration

Orchestration has become the predominant standard followed by enterprises for services composition, per se the most followed and applied standard is the Business Process Execution Language (BPEL), although other standards exists like the Business Process Modeling Notation (BPMN), which is the core enabler of Business Process Management (BPM). Both standards specify business rules and the order under which Web services interact to carry out a systems functionality. This section briefly reviews fault tolerance approaches that have checkpointing mechanisms as main core in the context of BPEL and BPMN. Firstly, we begin by reviewing many works that have local recovery as their main strength.

The first time someone implied that checkpointing was a suitable option for Web services was presented in [31], where Dialani et al propose an infrastructure and claim that it is transparent to Web services. Such

solution mounted on top of the Web services protocol stack, considering checkpointing based on message logging to restore and/or rollback a single Web service. Yet, the authors also argue that their proposal needs small modifications to recover globally. Nonetheless, such work is descriptive, yet it suggests the use of a local fault manager and global fault manager.

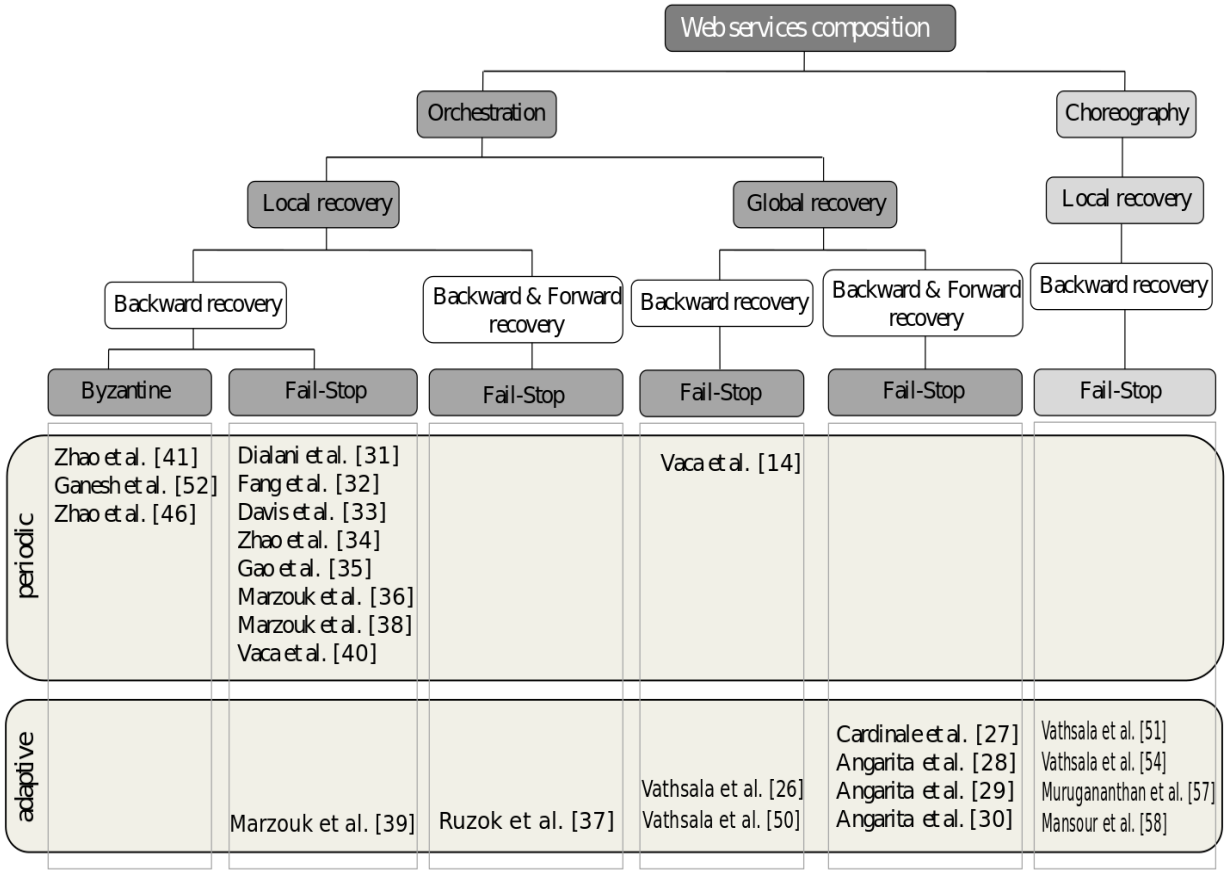

Fig. 1. Fault tolerance for composite web services: A taxonomy.

Davis *et al* patented a checkpointing technique for long running Web services [33]. It ensures the survival of Web services when an application server crashes or restart events occur. As stipulated by the author, the Web service state can be "revived" in response to a restart event. Yet, this patent considers individual Web services. They patent the idea that a checkpoint processor can be configured for coupling to individual Web services through a Web services engine. This processor is in charge of running the logic programmed to store and restore the corresponding check-pointed data from each of the failed Web services. Additionally, it manages the corresponding cleaning actions like removing checkpoint data that is no longer needed.

Fang *et al* present a framework called Fault Tolerant SOAP (FT-SOAP) based on previous integration middleware as is CORBA [32]. For interoperability reasons, at the time of writing, the authors examine two implementation approaches: one for SOAP's intermediary, and the other for Axis handler. Checkpointing is based on a logging mechanism which logs incoming requests and checkpoints critical states periodically for backups. However, for the intermediary approach, we found disadvantages like incompatibility, between their SOAP based logging service and other ones that do not implement such service, leading to an inconsistent state after a service is recovered form a crash. Another disadvantage is that, while checkpointing, the primary service is temporarily suspended until checkpointing is completed. Nevertheless, state checkpointing has a great impact on performance. Clients can experience delays while making an invocation to the primary Web services, because of checkpointing its states to its backups.

Wenbing Zhao presents a fault tolerance framework using replication as the main technological approach [34]. Such work tackles the Web service server side replicating *3f + 1* each client's incoming request. The author proposes to periodically perform garbage collection as not all replicas must be saved all the time. So, when the garbage collection performs its corresponding actions, so does the checkpointing mechanism.

Rukoz *et al* illustrate how a checkpoint mechanism can effectively be represented using Petri-Nets [37], providing a fault tolerant recovery scheme. Rukoz et al propose a three layer architecture: execution engine, engine thread and the actual Web services, located in the third tier. The execution engine manages the compensation order in case of failure. The engine thread runs a thread; for each peered Web service, it manages the execution control. If Web services fail, their approach is able to monitor and continue execution of the non-fail Web services as far as possible and then resume their execution from the last checkpoint.

Migration and replacement and/or rollback are found in the literature as an attractive way of guaranteeing Web services orchestration fault tolerance as found in [36, 38, and 39].

Marzouk *et al* achieve strong mobility defined as "enabling a running application component to be migrated from one host to another and to be resumed at the destination host starting from an intermediary execution state called checkpoint" by means of source code transformation [36]. They propose transformation rules in order to take checkpoints periodically. They also present three main transformation code aims: to maintain its updated state, to capture and to save the state when a checkpoint position is reached, and to load a checkpoint and to resume the execution starting from it.

Marzouk et al stipulate that self-adaptively is needed for applications under highly dynamic environments where applications components fail, or sometimes when performance degradations exists causing QoS' degradation [38]. They identify that other works focus on the unavailability of composite Web services and often use substitution for recovering, causing high overhead. Because other works do not use checkpointing, they have to restart all the orchestration. The authors discuss that their approach pursues the self-healing property; in case of failure, the failed process is migrated to a different server, and in case of a QoS violation, a subset of running instances may be migrated to a new server in order to decrease the initial host load. Marzouk et al. offer a flexible solution at runtime; the checkpointing policy dynamically changes, for instance, whenever the execution context changes an execution manager decides whether to change the checkpointing policy. Nevertheless, a recovery state is built after synchronizing all flow branches. This permits saving a consistent checkpoint. Yet, to our knowledge using synchronization for constructing a consistent checkpoint makes this approach expensive because of the barrier imposed from synchronizing; hence, this solution is slow and lacks concurrency.

The most complete work from Marzouk *et al* can be found in [39]. They present both the transformation rules and the aspects for strong mobility. They also illustrate that checkpoints can be forced based on policy-oriented techniques. Checkpointing techniques allow saving the state of an orchestration process and roll back to the last checkpoint taken; upon a failure and by making use of aspects, source code transformation rules and strong mobility only the non-executed code will be resumed and executed. In this work, the authors also take into account the quality of service (QoS), they do so by determining the checkpointing interval based on Markov chains and considering the required QoS of the mobile Web services. This is a sophisticated proposal and the authors present transformation rules, an adaptive dynamic computation of the checkpointing interval, and the selection of the mobility techniques. They use synchronization of parallel branches executed within a BPEL process and their work does not intend to build consistent global states from interacting business processes. Instead, they are able to build such from a single Web service within the BPEL process.

Varela *et al.* argue that companies need to intercommunicate exchanging information between business

logics, thus deciding to deploy what is called Business Process Management System (BPSM) [40]. BPSM helps to automate business processes, but in this context, systems are error prone and cannot guarantee a perfect execution over time. Therefore, a new paradigm called Business Process Management (BPM) arises. It is defined as a set of concepts, methods and techniques to aid the modeling, design, administration, configuration, enactment and analysis of business processes. For the business processes life cycle, the BPM paradigm follows diverse stages: design and analysis, configuration, enactment and diagnosis; however, each stage may introduce different fault kinds. For companies a way to gain dependability in early design stages is indispensable, promoting the reduction of possible faults and risks. In this work, the authors propose to follow traditional or classic fault tolerant ideas such as replication and checkpointing, among others, focusing on the service-oriented business processes context. However, such approach requires the introduction of extra components (sensors) into the business process design, extra time to check each sensor, and the recovery of business process service in rollback.

Table 1. Checkpointing for Local Recovery of Web Services

Reference	Global or Local Recovery	Byzantine or Fail-Stop Faults	Periodic or Adaptive Checkpointing	Backward and/or Forward Recovery	QoS-Awareness	BPEL or BPM
[31]	Local	Fail-Stop	Periodic	Backward possible Forward	----------	----------
[32]	Local	Fail-Stop	Periodic	Backward	----------	----------
[33]	Local	Fail-Stop	Periodic	Backward	----------	----------
[34]	Local	Fail-Stop	Periodic	Backward	----------	----------
[35]	Local	Fail-Stop	Periodic	Backward	----------	----------
[36]	Local	Fail-Stop	Periodic	Backward	----------	BPEL
[37]	Local	Fail-Stop	Adaptive	Backward and Forward	----------	----------
[38]	Local	Fail-Stop	Periodic	Backward	----------	BPEL
[39]	Local	Fail-Stop	Adaptive	Backward	Check-pointing based on QoS	BPEL
[40]	Local	Fail-Stop	Periodic	Backward	----------	BPM

Table 1 summarizes works that are based on checkpointing and are relevant for recovering a single Web service. Thus, it concerns local recovery.

Secondly, we review all works that carry out global recovery, where all participants, in this case Web services, must build and recover from a consistent global snapshot of the system in play. As an example, one can find that not many works focus their efforts on this kind of solution [14], [26]-[30].

Varela *et al.* identified that while executing business processes, they are susceptible to intrusion attacks, which can be the cause of severe faults [14]. Fault tolerance techniques tackle such issues, decreasing risk of faults, and are therefore more dependable, with the aim of achieving dependability before business processes automation. The authors claim that fault tolerance techniques can be applied in order to resists faults related to integrity attacks. Varela and Martinez proposed OPUS, a framework with many capabilities, developed following the Model-Driven Development (MDD) and the Model Driven Architecture (MDA). This framework has four layers: *Modeling, Application, Fault Tolerance and Services*. It is the Fault Tolerance layer which is based on checkpointing and rollback recovery. However, the authors do not mention which checkpointing mechanism they use. Often times new and improved checkpointing protocols are proposed in the literature. We believe that the recovery overhead time can be reduced by making use of such improved protocols.

Vathsala *et al* propose a way of building global checkpointing of orchestrated Web services [26]. To achieve such, they make use of a checkpointing policy. The authors contemplate a global set of checkpoints

in order to avoid expensive re-invocation of Web services that are synchronous, and therefore sequentially executed. To generate this global set, the authors compute all possible sequence of calls for an orchestrated Web service. They introduce the notion of Call-based checkpointing for Web services, thus they employ a set of checkpointing policies. These policies identify the calls within Web services; for instance a one way request will checkpoint its state for further use later. Nevertheless they tackle only one instance of the orchestration process, and do not take into account interaction among multi-party orchestration processes.

Cardinale *et al* propose a checkpointing approach using colored Petri nets [27]. This work is oriented towards Transactional Composite Web Service (TCWS), which present an atomicity property; such statement establishes an all-or-nothing behavior. In case of failure, their approach relaxes the aforementioned property to a something-to-all property. This solution encompasses both forward and backward recovery. This is because a snapshot is taken in by an advanced execution state; however, it must first give a partial result or return something to the user. Then for the user to get all later, a possible restart of the TCWS from the last snapshot is executed to complete the result. The main advantage of this work is that checkpoints are only taken in case of failure, therefore the authors claim that they do not increase the system overhead while the execution is free of failures.

In works [28]-[30] Angarita *et al* present a runtime decision-making model that chooses which recovery strategy is best suitable for a Web service within the execution of a Composite Web Services (CWS). The strategies include retry, compensation or checkpointing. In particular [28] presents a preliminary model to select the best recovery strategy in terms of impact on the CWS QoS. The authors extend their work to take into account more QoS criteria to obtain a self-healing model [29], presenting also the impact that different recovery strategies have on QoS and mention that their model chooses the best recovery strategy. Regarding checkpointing, techniques can be implemented to relax the all-or-nothing transactional property and still provide fault-tolerance, allowing users to have partial results and resume the execution later. Finally, in [30] Angarita et al. focus their efforts on providing a general model to support CWS executions, while maintaining required QoS and providing dynamism regarding the selection of fault-tolerance strategies. For all their works, they consider the dynamism of CWS execution, and the QoS's CWS during failure-free execution. Their global solution recovers the entire CWSs. The most recent aim for this kind of solution is to be integrated within dynamic CWS executions while maintaining the required QoS in presence of failures; such solution is automatic and distributed. Fault-tolerant CWS execution is based on transactional properties.

Vathsala *et al* aim at providing a way to make Web services orchestration resilient to faults [50]. They propose an adaptive checkpointing policy named *"Call Based Checkpointing of Orchestrated Web Services"*. This policy adapts depending on the mean time between failures and the prediction execution time, a comparison is made and depending on the type of operation carried out during a certain time of the executed orchestration, it decides whether or not to take a checkpoint. Additionally, it reduces the amount of checkpoints. One of the main advantages of this work is that, upon a failure, the entire system does not need to be repeated from the beginning. When Web services within the Web services composition become idle, the latest local checkpoint becomes the global checkpoint of the composed application; and the call-based global checkpoint is defined as a set of latest local checkpoints of each of the Web services that are active during the call. Upon a failure, the application rolls back to the latest global checkpoint and all messages replayed form the message logs. Execution continues without re-invoking the finished constituent Web services.

Table 2 summarizes works that are based on global recovery or the overall system recovery that implies using checkpointing mechanisms.

Table 2. Checkpointing for Global Recovery

Reference	Global or Local Recovery	Byzantine or Fail-Stop Faults	Periodic or Adaptive Checkpointing	Backward and/or Forward Recovery	Composite Web Services Approach	QoS awareness
[14]	Global	Fail-Stop	Periodic (using integrity sensors)	Backward Recovery	BPM	----------
[26]	Global	Fail-Stop	Adaptive	Backward	----------	----------
[27]	Global	Fail-Stop	Adaptive	Backward and Forward	Petri Nets	----------
[28]	Global	Fail-Stop	Adaptive	Backward and Forward	Graphs	Execution Time
[29]	Global	Fail-Stop	Adaptive	Backward and Forward	Graphs	Execution Time, Price, Reputation and Transactional properties
[30]	Global	Fail-Stop	Adaptive	Backward and Forward	Graphs	Execution Time
[50]	Global	Fail-Stop	Adaptive	Backward	----------	----------

Now we present works that consider a more troublesome kind of faults, specifically those known as Byzantine faults.

Byzantine faults are arbitrary and different users can experience diverse behavior of the system in play; they are more troublesome than fault-stop. These are only considered by few works [41], [46], and [52] for composite Web services; they implement replicas and fault tolerance mechanism and use checkpointing.

Marimuthu and Gopal consider Byzantine fault tolerance based on replication [52]. This kind of works were not feasible due to its runtime efficiency until the introduction of the work presented by Castro and Liskov [53]. Marimuthu and Gopal describe an asynchronous protocol that combines failure masking with imperfect failure detection and checkpointing; however, no implementation detail or performance evaluations are carried out regarding the checkpointing mechanism. This solution encompasses individual requests/responses made to Web services and replicates them $2t+1$ to mask Byzantine faulty ones; however, this solution does not consider global recovery of the overall system.

In the works [41], [46] Wenbing Zhao presents a fault tolerance framework capable of dealing with Byzantine faults, and not only crash faults. It does so by presenting a framework called BFT-WS that operates on top of SOAP for interoperability reasons and it is based on Castro and Liskov's BTF algorithm for efficiency. The author argues that this framework can overcome Web Services Reliable Messaging (WS-RM) drawbacks. In addition, BFT-WS is backward compatible with WS-RM, and when there is no need to replicate Web service it can run with the default WS-RM. Byzantine fault tolerance is achieved by replicating the server and executing in the same order of all replicas. Regarding checkpointing, it is used for garbage collection, where each replica periodically takes a snapshot of its state. The author adds two additional operations for checkpointing and recovery, namely, get state and set state. In order to update checkpoints while running the BFT algorithm, when a new checkpoint becomes stable, the previous ones along with all the control messages prior to the checkpoint are garbage collected. State restoration is also considered, for instance, when a slow replica has fallen too far behind. Finally, the authors present evaluation of their BFT-WS and claim that it has a low overhead compared to the complexity of this kind of solution. The main difference between [41] and [46] is that the former and most complete work supports multi-tiered Web services and transactional Web services, while the latter only considers single Web services.

Table 3 summarizes works that are most relevant for Byzantine faults which rely on checkpointing.

Table 3. Checkpointing for Byzantine Faults

Reference	Global or Local Recovery	Byzantine or Fail-Stop Faults	Periodic or Adaptive Checkpointing	Backward and/or Forward Recovery
[41]	Local	Byzantine	Periodic	Backward
[52]	Local	Byzantine	----------	----------
[46]	Local	Byzantine	Periodic	Backward

3.2. Fault Tolerance Techniques for Choreography

This section briefly reviews checkpointing based fault tolerance approaches in the context of Web services choreography and discusses their advantages and drawbacks. Composite Web services create complex business processes; however, they are more collaborative than orchestration. Web services are usually provisioned over the unreliable Internet, and are therefore susceptible to faults, so they must adopt fault tolerance techniques. Nevertheless, in spite of these research challenges, there has neither been much involvement from researchers, nor has it been tackled by the industry. To make Web services resilient to faults, Vathsala and Mohanty propose recovering Web services, by means of saving checkpoints in message logging [51], considering that only the failed Web services roll back, and it does not cause a chain of reactive services to rollback. Vathsala and Mohanty perform checkpointing of choreographed Web service at three different development stages: design time, deployment time and at runtime.

- At design time, they use the choreography document and introduce checkpoint locations at places where non-repeatable actions take place [54].
- At deployment time, they consider Web services non-functional requirements, such as QoS (response time, reliability, cost of service) and other quantities, like checkpointing time and message logging time.
- At runtime, the authors in a near future will dynamically predict QoS values and dynamic composition of Web services. Therefore, they need response time prediction as presented in [55, 56].

Vathsala et al identify the most appropriate checkpointing locations by means of their model, where they model the choreography composition as a set of interaction patterns. An introduction to this approach can be found in [20] and details can be found in [54]. Therefore, they stipulate that by making use of QoS values of services, they always met the execution times and cost of service constraints. To show the validity of their approach, they compare checkpointing Web services at design time and deployment time. They also aim for the minimum number of checkpoints during failure free execution, therefore, resulting in minimum overhead.

Muruganantham *et al* tackle Web services choreography based on an automatic checkpoint algorithm [57]. Their approach firsts locates Web services semantically or based on semantic search. As a second step, the Web services choreography is composed using AND/OR operators. As third step, they develop an auto checkpoint algorithm. Checkpoints are used to mark Web services, if such is executed successfully then the choreography moves to the next operation; otherwise, it restarts from a previous checkpoint. However, this work is merely descriptive and no further details are given. It only presents the system architecture and the rollback-recovery concept to enhance reliability.

Mansour and Dillon propose a new model for Web services modeling the error arrival time as a function of the workload of the server [58]. In this work, checkpoints are generated only when the broker realizes that the acceptance testing mechanism is deemed as unacceptable for a Web service of the composite Web services assembly. Checkpoints are associated with initiating a Web service and completion of a Web service. This work considers design errors, hardware server errors and channel transmission errors. The authors

are aware that the broker constitutes a single point of failure, to deal with it, they use Triple Modular redundancy and N-version Programming. Web services choreography is represented by a graph, each node represents a Web service and edges are placed between interacting Web services i.e. *i* to *j*. This is done sequentially, meaning that *j* is executed right after *i* within the choreography. Using acceptance testing based on positive or negative values of the quantity $E(i, j) = M - R - t$ checkpoints are placed or rollback is executed, where M is the maximum recovery time, R is the actual recovery time and t is the execution time of service *j*. For instance, if $E(i, j)$ is negative a checkpoint is inserted between Web service i and j and so on for the next sequential task defined in the choreography.

Table 4 summarizes works that use checkpointing as their recovery technique within Web services choreography.

Table 4. Checkpointing for Choreographies

Reference	Global or Local Recovery	Byzantine or Fail-Stop Faults	Periodic or Adaptive Checkpointing	Backward and/or Forward Recovery	QoS-Awareness
[51]	Local	Fail-Stop	Adaptive	Backward	QoS Checkpointing policy
[54]	Local	Fail-Stop	Adaptive	Backward	----------
[57]	Local	----------	Periodic	Backward	----------
[58]	Local	Fail-Stop	Periodic	Backward	----------

4. Discussion and Open Challenges

For this literature survey, we found many papers that describe diversity of approaches (recovery modes, type of faults considered, etc.) highlighting the importance of standardization, since there is no common solution incoming from the authors. For instance, fault tolerant mechanisms should be a means by which composite Web services recover: partially, totally, globally and or locally. Not until there is a common agreement among researchers the applicability of the approaches in industry will be hindered. Although all fault tolerance architectures agree on where to place the fault tolerant capability within the Web services protocol stack, a common problem found is that these solutions require special analysis models or familiarity with mathematical models (Petri Nets, Markov Chains).

Despite the fact that taxonomies that classify faults exist, we found in our literature survey that the treated faults are missing in many works. Therefore, in this survey we propose a novel taxonomy for Web services composition based on the currently most used standards, such as choreography and orchestration. Not many works are currently developed for choreography by means of checkpointing. Those that exist, generate checkpoints automatically and in case of detecting a failure, only one Web service applies a rollback recovery strategy. Nonetheless, an open opportunity for Web services choreography is global recovery instead of local or individual recovery, contemplating non-functional requirements, such as QoS.

Only few works deal or contemplate QoS while checkpointing Web services composition. In general, in this survey, checkpointing can be carried out periodically; nonetheless, this can lead to inconsistent states or it can be carried out in an adaptive manner. We consider this is the best way of doing so. Another noteworthy fact is that only one work considers the checkpointing interval as traditional checkpointing mechanisms for distributed systems do. A confusing fact is that most works imply they use checkpointing as means for Web services and Web services composition fault tolerance, but fail to mention which mechanism they use, and whether it is a distributed or a centralized solution. More work needs to be carried out for both orchestration and choreography leveraging quasi-asynchronous checkpointing advantages, such as asynchronous execution and de-centralized nature.

Future trends indicate that there will be a time when choreographies interact against other choreographies. Orchestrations will need to communicate or intercommunicate with other orchestrations and possibly a combination of these aforementioned technologies. Therefore, new and difficult challenges can arise while adopting fault tolerance based on checkpointing mechanisms, which take into account QoS, and checkpointing interval, oriented towards rollback recovery for emerging trends.

Other open challenges include not taking checkpoints at regular intervals of time or periodically, since doing so can revert the system to an inconsistent state. Instead they could depend on the interaction and quality of service among Web services both for existing trends like BPEL and choreographies, as well as for new trends such as interactive BPEL processes. In addition, it is a well-known fact that checkpointing mechanisms have a correlation between recovery overhead, in case of failures, and checkpointing overhead, in case of system failure-free execution. The question that remains as an open challenge involves the quality of service of the involved business processes. Only one work stipulates that they do not incur an overhead during the failure-free execution. Such solution only checkpoints when needed (when faults happen). More challenges include: handling execution programs, partial failures, machine crashes, and conserving data coherency across machines in such situations.

One last note, composite Web services are characterized by their loose coupling, distributed data and distributed components, as well as their asynchronous interactions. However, these are not completely supported by current works. For example, imposing a barrier to synchronize flows inhibits asynchronous interactions among components, which in turn, slows down the system. Another clear example is when works report periodically-saved local checkpoints; this may lead to global inconsistent states. The design of fault tolerant mechanism for Web services composition based on checkpointing presents the following open questions:

To be efficient the following questions arise:

- How often and when must the checkpoints be taken? Most of the works take checkpoints periodically and only some do it adaptively according the system behavior.
- Where or who shall take the checkpoints? Most works rely on proprietary models and there is not a common agreement on such topic.

To be consistent the following question arises:

- Which are the properties that must be satisfied between checkpoints to establish consistent global snapshots? None of the aforementioned works perform a formal verification that they actually rollback to consistent states.

To accomplish the distributed and asynchronous nature of a composite Web service the following question arises:

- Which is the most suitable checkpointing technique that better adapts to the nature of Web services composition? Most of the works are based on checkpointing for message logging; however, it has its disadvantages like possible rollback to inconsistent system global state. None of the previous analyzed works is based on the quasi-synchronous checkpointing technique. Such technique must be explored for fault tolerant Web services composition to leverage their distributed and asynchronous inherent characteristics.

5. Conclusion

Although sophisticated solutions exist that merge checkpointing and Web services paradigms and try to tackle Web services fault tolerance, most of these solutions focus merely on a single Web service instance; for example, a Business Process Execution Language (BPEL) process and apply substitution, transformation rules, migration or combinations of diverse approaches. An identified open challenge is to establish

consistent global snapshots from check-pointed data for emerging trends such as: interactive business processes (BPEL), interactive choreographies and interactive orchestrations conclusion section is not required. Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions.

We conclude that the price for achieving fault tolerance in some cases affects the scalability of the system or has a negative impact on the systems performance, and oftentimes dynamical environments are not taken into account. In this sense, an open challenge involves participants constantly entering and leaving the system. Thus, more work has to be conducted to guarantee Web services composition fault tolerance; which will provide a better quality perceived by the end user and organizations.

Finally, all the above works have been proven worthy of consideration, nonetheless, it would be a milestone for checkpointing mechanisms for Web services composition fault tolerance based on checkpointing if diverse work groups can establish open standards, for instance, regarding when to checkpoint, optimal checkpointing intervals, and QoS driven policies for checkpointing.

Acknowledgment

Part of this work has been supported by the Consejo Nacional de Ciencia y Tecnologia (CONACYT) from Mexico.

References

[1] Charles, J. P. (2016). *Web Service Composition*. Springer.

[2] Francisco, C., Matthew, D., Rania, K., William, N., Nirmal, M., & Sanjiva, W. (2002). Unraveling the web services web: Anintroduction to soap, Wsdl, and Uddi. *IEEE Internet Computing*, *6(2)*, 86-93.

[3] Angel, L. L., Florian, D., & Boualem, B. (2016). Web service composition: A survey of techniques and tools. *ACM Computing Surveys (CSUR)*, *48(3)*.

[4] Mustafa, B., Mark, H., & Youssef, H. *et al*., Testing web services: A survey. *Department of Computer Science*, Kings College London, Tech. Rep. TR-10-01, 2010.

[5] James, P. (2005). How BPEL and SOA are changing web services development. *IEEE Internet Computing*, *9(3)*, 60–67.

[6] John, F., & Joey, F. (2012). The service-oriented media enterprise: SOA, BPM, and web services in professional media systems. CRC Press, 2012.

[7] Ahmed, A., & Bernhard, H. (2010). Quality of service attributes in web services. *Proceedings of the* 2010 *Fifth International Conference on Software Engineering Advances*.

[8] Moody, A., Bronevetsky, G., Mohror, K., & Supinski, B. R. D. (2010). Design, modeling, and evaluation of a scalable multi-level checkpointing system. *Proceedings of the 2010 International Conference for High Performance Computing, Networking, Storage and Analysis*.

[9] Ínigo, G., Ferran, J., Jordi, G., & Jordi, T. (2010). Checkpoint-based fault-tolerant infrastructure for virtualized service providers. *Proceedings of the 2010 IEEE Network Operations and Management Symposium (NOMS)*.

[10] Zheng, Z. B., & Michael, R. L. (2010). An adaptive qos-aware fault tolerance strategy for web services. *Empirical Software Engineering*, *15(4)*, 323–345.

[11] Zheng, Z. B., & Michael, R. L. (2012). Optimal fault tolerance strategy selection for web services. *Web Service Composition and New Frameworks in Designing Semantics: Innovations*.

[12] Zheng, Z. B., & Michael, R. L. (2013). Qos-aware fault tolerance for web services. *In QoS Management of Web Services*.

[13] Zheng, Z. B., & Michael, R. L. Selecting an optimal fault tolerance strategy for reliable service-oriented systems with local and global constraints. *IEEE Transactions on Computers*, *64(1)*, 219–232.

[14] Angel, J. V. V., & Rafael, M. G. (2010). Opbus: Fault tolerance against integrity attacks in business processes. *Computational Intelligence in Security for Information Systems 2010*.

[15] Chris, P. (2003). Web services orchestration and choreography. *Computer*, *36(10)*, 46–52.

[16] Aarti, K., Milind, K., & Meshram, B. B. (2011). Choreography and orchestration using business process execution language for SOA with web services. *International Journal of Computer Science Issues IJCSI*.

[17] Mohsen, R., Walid, F., & Claude, G. (2012). Web services compositions modelling and choreographies analysis. *Web Service Composition and New Frameworks in Designing Semantics: Innovations: Innovations*.

[18] Ajay, K., Nikolaos, G., & Valerie, I. (2013). QOS composition and analysis in reconfigurable web services choreographies. *Proceedings of the 2013 IEEE 20th International Conference on Web Services*.

[19] Quan, Z. S., Qiao, X. Q., Athanasios, V. V., Claudia, S., Scott, B., & Xu, X. F. (2014). Web services composition: A decade's overview. *Information Sciences*.

[20] Vathsala, A. V., & Hrushikesha, M. (2014). A survey on checkpointing web services. *Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*.

[21] Mariano, V. S., Saul, E. P. H., Luis, A. M. R., & Hatem, H. K. (2016). Fault tolerance approach based on checkpointing towards dependable business processes. *IEEE Latin America Transactions*, *14(3)*, 1408–1415.

[22] Ajay, D. K., & Mukesh, S. (2008). *Distributed Computing: Principles, Algorithms, and Systems*. Cambridge University Press.

[23] Ann, T. M. (2003). Web services: A manager's guide. Addison-Wesley Longman Publishing Co., Inc., 2003.

[24] Francisco, C., Yaron, G., Johannes, K., Frank, L., & Weerawarana, S. *et al.* (2003). Business process execution language for web services.

[25] Meiko, J., Nils, G., & Ralph, H. (2009). A survey of attacks on web services. *Computer Science - Research and Development*, *24(4)*, 185–197.

[26] Vathsala, A. V. (2012). Global checkpointing of orchestrated web services. *Proceedings of the* 2012 1st *International Conference Recent Advances in Information Technology*.

[27] Yudith, C., Marta, R., & Rafael, A. (2013). Modeling snapshot of composite WS execution by colored PETRI nets. *In Resource Discovery*.

[28] Rafael, A., Yudith, C., & Marta, R. (2013). Dynamic recovery decision during composite web services execution. *Proceedings of the Fifth International Conference on Management of Emergent Digital EcoSystems*.

[29] Rafael, A., Marta, R., & Yudith, C. (2015). Modeling dynamic recovery strategy for composite web services execution. *World Wide Web*, 1–21.

[30] Rafael, A., Marta, R., & Maude, M. (2015). Dynamic composite web service execution by providing fault-tolerance and QOS monitoring. *Service-Oriented Computing-ICSOC 2014 Workshops*.

[31] Vijay, D., Simon, M., Luc, M., David, D. R., & Michael, L. (2002). Transparent fault tolerance for web services based architectures. *In Euro-Par 2002 Parallel Processing*, 889–898.

[32] Chen-Liang, F., Deron, L., Fengyi, L., & Chien-Cheng, L. (2007). Fault tolerant web services. *Journal of Systems Architecture*, *53(1)*, 21 – 38.

[33] Davis, D. B., Tan, Y., Topol, B. B., & Vellanki, V. (2009). Checkpointing and restarting long running web services.

[34] Wenbing, Z. (2007). A lightweight fault tolerance framework for web services. *Proceedings of the*

IEEE/WIC/ACM International Conference on Web Intelligence.

[35] Le, G., Susan, D. U., & Janani, R. (2011). A survey of transactional issues for web service composition and recovery. *International Journal of Web and Grid Services*, *7(4)*, 331–356.

[36] Marzouk, S., Maalej, A., Bouassida, I., & Jmaiel, M. (2009). Periodic checkpointing for strong mobility of orchestrated web services.

[37] Marta, R., Yudith, C., & Rafael, A. (2012). Faceta*: Checkpointing for transactional composite web service execution based on petri-nets. *Procedia Computer Science*, *10(0)*, 874 – 879.

[38] Marzouk, S., Maalej, A., & Jmaiel, M. (2010). Aspect-oriented checkpointing approach of composed web services.

[39] Soumaya, M., & Mohamed, J. (2013). A policy-based approach for strong mobility of composed web services. *Service Oriented Computing and Applications*, *7(4)*, 293–315, 2013.

[40] Vaca, A., Gasca, R. M., Borrego, N. D., & Hidalgo, S. P. (2011). Fault tolerance framework using model-based diagnosis: Towards dependable business processes. *International Journal on Advances in Security*.

[41] Wenbing, Z. (2007). Bft-ws: A byzantine fault tolerance framework for web services. *Proceedings of the EDOC Conference Workshop Eleventh International IEEE*.

[42] Chan, K. S. M., Judith, B., Johan, S., Luciano, B., & Sam. G. (2007). A fault taxonomy for web service composition.

[43] Gupta, S., & Bhanodia, P. (2013). A fault tolerant mechanism for composition of web services using subset replacement. *Architecture*.

[44] Anne, I., & Daniel, P. (2014). A survey of methods and approaches for reliable dynamic service compositions. *Service Oriented Computing and Applications*, *8(2)*, 129–158.

[45] Jonathan, L., Shang-Pin, M., Shin-Jie, L., Chia-Ling, W., & Chiung-Hon, L. L. (2011). Towards a high-availability-driven service composition framework. *Service Life Cycle Tools and Technologies: Methods, Trends and Advances*.

[46] Wenbing, Z. (2009). Design and implementation of a byzantine fault tolerance framework for web services. *Journal of Systems and Software*, *82(6)*,1004–1015.

[47] Yin, J., Chen, H., Deng, S., Wu, Z., & Pu, C. (2009). A dependable ESB framework for service integration. *Internet Computing*, *13(2)*, 26–34.

[48] Louise, E. M., Melliar-Smith, P. M., & Zhao, W. B. (2007). Building dependable and secure web services. *Journal of Software*, *2(1)*,14–26.

[49] Richard, K., & Sam, T. (1987). Checkpointing and rollback-recovery for distributed systems. *IEEE Transactions on Software Engineering*, 23–31.

[50] Vathsala, A. V. (2011). Optimal call based checkpointing for orchestrated web services. International Journal of Computer Applications, *36(8)*.

[51] Vani, V. A., & Hrushikesha, M. (2015). Time and cost aware checkpointing of choreographed web services. *Distributed Computing and Internet Technology*.

[52] Gopal. D. G. (2011). A novel approach for efficient resource utilization and trustworthy web service. *International Journal of Computer Science and Security*, *5(2)*.

[53] Miguel, C., Barbara, L., *et al.* Practical byzantine fault tolerance.

[54] Vathsala, A. V., & Hrushikesha, M. (2014). Interaction patterns based checkpointing of choreographed web services. *Proceedings of the 6th International Workshop on Principles of Engineering Service-Oriented and Cloud Systems*.

[55] Vathsala, A. V., & Hrushikesha, M. (2012). Using hmm for predicting response time of web services. In *Proceedings of the CUBE International Information Technology Conference*.

[56] Vani, V. A., & Hrushikesha, M. (2015). Web service response time prediction using hmm and bayesian network. *Intelligent Computing, Communication and Devices*.

[57] Murugananthan, B., Vivekanandan, K., & Mondal, D. (2014). Rollback recovery approach for complex composite web services to enhance reliability of service. *Interantional Journal of Engineering Research and Technology*, *3(2)*, 2289–2292.

[58] Houwayda, E. M., & Tharam, D. (2011). Dependability and rollback recovery for composite web services. *Transactions on*, *4(4)*, 328–339.

Mariano Vargas Santiago received the master in research degree in information and telecommunications from Institut National des Sciences Appliquées, Toulouse, France, in 2013, and is a PhD student in distributed systems and software engineering at Instituto Nacional de Astrofísica Óptica y Electrónica, México. His current research interest are merging two widely used paradigms, as are checkpointing mechanisms and autonomic computing.

Saul E. Pomares Hernandez is a researcher in the computer science Department at the National Institute of Astrophysics, Optics and Electronics (INAOE), in Puebla, Mexico. He completed his PhD Degree at the Laboratory for Analysis and Architecture of Systems of CNRS, France in 2002. Since 1998, he has been researching in the field of distributed systems, partial order algorithms and multimedia synchronization

Luis A. Morales Rosales received the engineering degree in computer systems from Instituto Tecnológico de Colima, Colima, México, in 2001, and his PhD in computer science, in 2009, from Instituto Nacional de Astrofísica Óptica y Electrónica, Tonantzintla, Puebla, México. His current research interest are applications of distributed systems and intelligent computing.

Hatem Hadj Kacem is a researcher in the computer science at the Faculty of Economics Sciences and Management, University of Sfax, Tunisia. He completed his PhD Degree at the Faculty of Sciences and Techniques of Rouen, France in 2005. Since 2006, he has been researching in the field of theoretical computer science