A survey on Change Management in Service Based System using SOA

V. Kavitha and Dr.P. Shanmugapriya

1Research Scholar, SCSVMV University, Asst.Prof. Sri Sai Ram Engineering College,
2Assoc. Prof, SCSVMV University.

Received 7 June 2016; Accepted 12 October 2016; Available 20 October 2016

ABSTRACT
In the service-oriented computing paradigm, inter-organizational applications and information systems can be built upon services from different providers. Services are subject to changes required by the user, organizational and regulatory policies changes. The challenging problem of service change management has been studied actively in recent years. This paper provides an extensive overview of the current research on change management, Analysis of change and its impact in the context of service-oriented computing. First, classify service changes into two major types functional and non-functional changes. Then, we review the existing work on change management based on Service Adaptation, Change Analysis and Management in Service Compositions. In each category, the discussion is based on focusing changes, the proposed approaches to dealing with the change problems, and the change issues that remain to be solved.

KEYWORDS: Keyword1: Service-oriented computing Keyword 2: Change management Keyword 3 : Change analysis Keyword 4 : Change Impact Keyword 5: web services.

INTRODUCTION
Change management is a traditional problem and an essential part in the development and maintenance of software based systems, due to the evolution nature of computing programs [1]. Without being related to service-oriented computing (SOC), change management has been studied in a wide range of research areas: software engineering [2,3,4,5,6,7,8], distributed systems [9,10,11,12], database management systems [13,14,15,16,17], information systems [18,19], and workflow systems [21,22,23-30,31]. For example, the change management in database systems focuses on handling dynamic schema changes and evolution. Closely related to the change management in Web service-based systems, in the software engineering area, adaptation approaches to mediating mismatches between software components are proposed. In addition, for distributed systems and information systems, principles, and mechanisms for managing evolutionary changes introduced to system components and modules that are not expected at the design time exist. These efforts are valuable and can be leveraged to develop mechanisms for the Web service adaptation and the change management for Web service-based systems. However, they are insufficient for the reason that external specifications of Web services contain richer information than component interfaces. Moreover, various types of changes of Web services that belong to different service providers usually happen independently, which may affect cross-organizational collaborations building on these Web services. The workflow change management, which has been extensively studied since 1990s, is also related to the research on the change management for service-based business processes. The focus is put on handling workflow schema changes (static change) and the migration of workflow instances (dynamic change) when their schemas change. These results are useful for enabling BPEL.
business process flexibilities, handling service protocol changes and evolution. However, due to the dynamic nature of Web services and the complex relationships between services and business processes, these studies are inadequate to fully address the change issues and support the flexibilities for service-based business processes.

The SOC paradigm facilitates the low-cost and rapid composition of loosely coupled applications. Cross-organization applications can be built-up by integrating heterogeneous services from different providers. Due to various origins such as organizational and regulatory policies, environmental conditions, user requirements and technologies, services may need to change from time to time.

A service can change in many aspects including its interface, business protocol as well as QoS attributes. A change of a service interface and a business protocol may cause incompatibilities between interacting parities. Moreover, a complex service may be supported by multiple underlying business processes [32]; a change to the service usually requires changes applied to its supporting business processes and consequently to other services the same business processes support. In a word, a service change can have various types and different levels of impact on interacting services and associated business processes. Service-based applications and information systems will need to operate correctly despite of expected and unexpected changes related to services and business processes. Therefore, when a change happens, it is important to understand the type of this change, the change impact including the affected scope and how deeply it affects the entire system and the proper reaction to handle it. Due to the distributed and dynamic characteristics of services, change management is a challenging issue in service-based environments [33,34].

Reference architecture also has been developed for managing dynamic inter-organizational business processes which considers a set of functional and non-functional requirements for eSRA that comprises components on three refinement levels and their information exchanges. The collaborating parties must be able to protect their business secrets and the overall performance of an eSRA-compliant system Must not impede business collaboration. [35].

The end goal of the service change management is to realize change automation in the entire life cycle of service-based applications. We have observed theorems, guidelines, and approaches reported in the literature that deals with different change problems in the development and maintenance of service-based systems. The purpose of this paper is to provide an extensive survey of the existing research on change management in service-based environments. In this paper, we shall review the existing research from the following categories:

Service Adaptation:
refers to the ability of services to adjust themselves in order to overcome incompatibilities caused by changes in interacting services at the level of interface and/or external behavior. The main idea for service adaptation is about designing adapters/operators to mediate various types of differences between Web services. To generate adapters, a number of approaches are based on the major types of mismatches relating to service interfaces and/or business protocols; another branch of methods is based on formal modeling of services, such as Petri nets and state machines.

Process Flexibility:
Extensions to the BPEL specification are suggested in the literature to improve the flexibility of business processes. Requirements such as contextual information and QoS properties are also considered in some work in order to cope with changes relating to context and customer requirements.

Service Evolution:
is the process of changing services dynamically and consistently. In this branch of research, guidelines and mechanisms supporting Web service versioning control and evolving service specifications such as business protocols are suggested.

Changes Analysis and Management in Service Compositions:
Web services operate in highly dynamic environments. Participant Web services in a service composition, especially for long running service compositions, may change at any time, for example, operation of a Web service becomes unavailable. In the literature, we have seen several efforts that provide mechanisms for changes analysis and management in service compositions. The focus is on detecting, analyzing, propagating, as well as reacting to service changes.

This paper is organized as follows. In Sect. 2, we classify the types of changes that can happen to services into two categories as behavioral and non-functional changes. In Sect. 3, we review and summarize the methodologies, techniques, and approaches proposed in the literature for dealing with the various types of change issues. Finally, we conclude this paper in Sect. 4.
2 Changes in service-based environments:

Services can vary in functions ranging from checking a credit number to executing complicated business processes. Services are published, discovered, selected, and integrated based on their external specifications, which contain the information of signatures, message exchanging constraints and orders, QoS properties, and behaviors of services. Due to various reasons such as business regulations, application environments, and client requirements, services need to change from time to time. A service can change in many aspects of its external specification, such as data schemas of messages, operation granularity, message exchanging sequences, and operation existence etc., in order to engage in business collaboration.

The XML-based Web services specifications: WSDL and BPEL provide standards for describing the interface and behavior of a Web service. Apart from these industrial standards for service description, we have seen a number of modeling techniques: process algebras [36,37,40-50], automata [51,52,53,], and Petri nets [54-56] in the literature in order to provide formal semantics for the behavior of Web services. In this paper, abstracting from the modeling techniques of services, we categorize the changes that can happen to services into the following three broad categories:

1. Functional changes:

   The behavior of a Web service describes how it interacts with its clients by exchanging messages. For instance, business protocols and abstract BPEL processes are common means of specifying service behaviors. Therefore, behavioral changes include variations relating to service behaviors, for example, changing message orders in business protocols or removing activities in abstract BPEL processes.

2. Non-functional changes:

   Contains variations of QoS properties. In general, the QoS of a Web service includes a number of non-functional attributes such as privacy, reputation, usability, and execution price, which are important for the potential customers of this service [57]. Services having similar functions may have different QoS parameters. QoS properties are important criteria in forming service level agreements (SLAs) between collaborating business partners. QoS properties of a Web service may change at any moment. For example, a service provider can modify the execution price or the penalty rate of its web services at any time.

   A change can have various levels of impact: a service change may be restricted locally without any further impact on the entire system; it may also deeply affect the entire value chain and has ripple effects as well [58]. Thus, change detection and impact analysis are crucial and complex tasks in the change management for service-based applications and information systems. In the following section, we shall review the major work that have been done for dealing with change issues relating to service interfaces, behaviors, and non-functional attributes

3 Service change management:

   This section discusses the work done for dealing with service changes in the context of SOC. According to the above three research categories, we shall firstly review the major approaches and techniques for adapting service interfaces and protocols, which are important means for handling certain types of change problems. Then, we will discuss the approaches proposed in the literature for enabling process flexibility and service evolution. Finally, we will focus on the work for change analysis and management in service-based systems.

3.1 Service adaptation:

   Changes can cause incompatibilities between interacting services. For example, a service may evolve to a new version which may have different operation invocation sequences compared with the old version. Then, there may have incompatibilities between the service and its clients. Although current Web service standards such as SOAP and WSDL can solve the compatibility issue between Web services at the lower level of abstraction, that is, messaging, incompatibilities still exist at higher levels of abstraction, that is, service interfaces and service behaviors. Service adaptation refers to the capability of changing a service itself in order to interact with other services [56]. Service adaptation is an important means to achieve interoperability among interacting services when changes happen to any of these services.

   Current research on service adaptation mainly concentrates on adapting service interfaces and business protocols. Adaptation at the interface level is about adjusting inter-faces in order to overcome differences and incompatibilities among interacting services. The incompatibilities at this level include mismatches between signatures such as parameter mismatches and operation mismatches etc. A service protocol describes the desired message exchange sequences between the service and its clients [9]. Adaptation at the protocol level is about mediating the various types of differences among service protocols. There are two major types of approaches to adapting services: adaptation based on mismatch patterns and adaptation based on formal semantics.
3.1.1 Service adaptation based on mismatch patterns:

Abstracting from notations for describing service interfaces and behaviors, Benatallah et al. [8] classify incompatibilities among services into the following types/patterns as signature mismatch (S), message order mismatch (O), extra/missing message mismatch (E/M), and message split/merge mismatch (S/M). Using adapters is a typical and effective means for solving differences among Web services. A number of approaches have been proposed for designing service adapters for handling (some of) these various types of differences.

Benatallah et al. [8] propose an innovative approach: designing patterns for adapters based on a set of mismatch patterns for service interfaces and protocols. This research provides a general guideline for dealing with service adaptation based on classifications on service differences. Motahari-Nezhad et al. [52,53] present semi-automated mechanisms for detecting and designing adapters to mediate some types of incompatibilities between services. In [52], algorithms for detecting and mediating the various types of mismatches are devised. For deadlock caused by message order mismatch, mechanisms for resolving the deadlock are provided based on a set of criteria. As an extension to their previous work, in [53], the authors provide algorithms for mediating S/M mismatch considering the related protocol information. Based on the mismatch patterns, Kongdenfha et al. [36] propose an aspect-oriented framework to handle differences between an internal service implementation and standard external specifications at the interface and protocol level. The business logic is treated as the main concern, and the adaptation logic is specified as crosscutting concerns. The authors design a set of templates to handle the mismatch types: S, O, E/M, and S/M. Each template comprises a point-cut and an advice: the former is specified as queries over a business process execution, while the latter is described by the BPEL specification to express the actions to be performed when handling a specific type of mismatch. A prototype tool is developed to support the template instantiation and execution. The mechanisms for designing adapters for coping with some of the mismatch types with special consideration, for example, minimal adapters, have been suggested. Different from the above work, Ponnekanti and Fox investigate the interoperability of independently evolving Web services in Web-based applications. The goal is to enable substitutions of functionally similar Web services that are derived from a common base. They identify four types of incompatibilities between Web services as structural incompatibility, value incompatibility, encoding incompatibility, and semantic incompatibility. The authors focus on the structural and value incompatibility referred to as SV-incompatibility.

The SV-incompatibility is further categorized into five types as missing methods, extra fields, missing fields, facet, and cardinality incompatibilities. The combination of static and dynamic analysis can determine SV-incompatibility between Web services automatically. The interoperability between Web services then can be realized by semi-automatically generated cross-stubs.

In general, adapters are implemented as middleware components that mediate two Web services in a service environment. From the above description, we can observe that automatic adapter generation for mediating the various types of incompatibilities is a challenging task, especially for semantic incompatibilities. Relying on types of differences, adapter templates, and mismatch detection algorithms, the generation of adapters for coping with a specific incompatibility requires human intervention and input. Efforts are still required to provide automatic adapter generation mechanisms that can realize seamless system integration based on Web services.

3.1.2 Service adaptation based on formal semantics:

To facilitate automated analysis of service compatibility, business protocols/service behaviors are described by formal semantics such as finite state machine, Petri nets, and process algebra in a number of studies. Based on formal semantics, approaches have been proposed to automatically analyzing compatibilities between interacting Web services and to generate adapters for overcoming incompatibilities. The major work in this area into three sub-problems: service compatibility, protocol adaptation, and behavioral interface adaptation. In the service compatibility category, the focus is on defining, detecting, and verifying the compatibilities between interacting services based on formal modeling tools. The protocol adaptation aims at automatically generating service adapters based on formal modeling notations. The behavioral interface adaptation focuses on the problem of adjusting behavioral interfaces Bordeaux et al. [12] describe a service as a labeled transition system, which consists of a set of states, transitions between states and actions (receive/send messages). Based on this notation, the authors define three types of behavioral compatibility. Lehmann et al. [45,46] and Martens et al. [47,48] both translate BPEL processes into some type of Petri nets so that interactions between two BPEL processes can be analyzed formally and automatically. In addition to compatibility checking, Lehmann et al. also provide a set of tools: BPEL2oWFN and Fiona, for generating service adapters. Martens et al. focus on formally defining and automatically checking compatibility and equivalence of service behaviors. Still for behavioral compatibility checking and verification, Foster et al. [24] transform BPEL processes into a so-called finite state process notation. Pongee et al. concentrate on automated analysis of compatibility between Web service protocols with timing constraints. Service protocols are defined in finite state machines and enriched with timing abstractions. Types of compatibility and replace ability, for example, partial/full compatibility, are identified based on the formal notation. A set of operators are provided for analyzing the identified types of compatibility and replace ability of timed protocols.
Formal semantics are also advocated for automated adapter generation by some efforts. Normally, these work assume mappings at the interface level are provided. With the help of LOTOS process algebra, Mates, Poizat, and Salain [49] provide a series of tools to support adapter generation. Brogi and Popescu [13] translate targeted BPEL processes into YAWL workflows. Then, an adapter for the YAWL defined workflows is built, which is also in the form of YAWL workflow. The newly created adapter is tested for deadlocks and deployed as BPEL process. Using fuzzy logic, Pernici et al. [57] focus on the problem of selecting suitable adaptation strategies for QoS changes. The key idea is to define fuzzy parameters for the QoS property descriptions of Web Services. Thus, partial satisfaction of imprecise requirements is allowed.

Dumas et al. [20] consider the adaptation of behavioral service interfaces which contain not only messages but also the order and constraints between these messages. A behavioral interface is described by a set of traces over an alphabet made up of communication actions. The authors define six algebraic transformation operators to mediate the different types of interface mismatches. An operator takes a service interface as input and generates an interface as output. Also for behavioral interface adaptation, Queering et al. [55] model an interface as a symbolic transition system. Incompatibilities between client and provider can be calculated based on the formal model. Steps for mediating mismatches are given with the help of interface mapping trees and mismatch lists. From the above discussion, we can observe that formal modeling tools/languages provide a powerful means for formally defining and verifying service compatibilities and also automatically generating adapters for overcoming differences between Web services/BPEL processes. These approaches require Web services/BPEL processes to be transformed into formal notations in the first place before any analysis and adaptation can be carried out. We have seen most of these approaches are evaluated by examples and scenarios with good performance. However, the scalability of these approaches is not addressed.

3.2 Process flexibility:

Without being related to SOC, business processes flexibility has been extensively studied in the context of workflow systems [1,28,58,69,78,80]. In [88], Barbara Weber, proposed 18 change patterns and seven change support features which – in combination – allow for assessing PAIS change frameworks. The introduction of change patterns complements existing workflow patterns and allows for more meaningful evaluations of existing systems and approaches, particularly if flexibility is an issue. In combination with workflow patterns the presented change framework will enable PAIS engineers to choose the process management technology which meets their flexibility requirements best. In service-based environments, business processes can be created by integrating widely available Web services from multiple business partners.

The distributed and dynamic characteristics of business processes in the context SOC demand higher requirements of flexibility compared with traditional workflow processes in order to cope with the various changes and uncertainties arising from highly dynamic environments. In this subsection, we discuss the main research on process flexibility in the context of SOC from the perspective of business process management. We shall review this branch of work as follows from the three major categories: extensions to BPEL, contextual changes, and QoS changes.

Extensions to BPEL BPEL is the de facto industry standard for orchestrating Web services offered by different service providers into business processes. Extensions to the BPEL specification have been proposed in the literature for improving the flexibility and adaptability of business processes. Some works [2,16,31,36] use the aspect-oriented programming technique to extend the BPEL specification. For instance, in [16], Charfi and Mezini employ the aspect-oriented programming technique to address crosscutting concerns such as logging, persistence, and security in BPEL processes. They propose the AO4BPEL as an extension to the BPEL language. In AO4BPEL, the business logic is treated as the main concern while crosscutting concerns such as data validation and security are specified using aspects. Koning et al. [37] propose a modeling language: VxBPEL, an extension to BPEL, which aims to model variability in Web service compositions based on variation points and variants. A variation point refers to a part of a system that can change. A variant is a change option that can be applied to a variation point. Three types of variability are modeled by the proposed VxBPEL as service replacement, service parameters, and system composition. Agarwal and Jalote [2] extend the BPEL specification so that users can specify non-functional requirements of Web services. In [51], Mietzner and Leymann introduce the notion of variability descriptor to customize process based and service-oriented applications. A variability descriptor specifies variability points and dependencies between variability points. Variability descriptors are transformed into BPEL processes. Multiple variability descriptors can be assigned to an application template to realize different adaptation according to different customer requirements. Zeng et al. advocate a policy-driven approach for exception management in BPEL processes. The key idea is the separation of the business logic and the exception handling policies. The specified exception handling policies are integrated with business logic at runtime to generate exception-aware process schemas.

Contextual changes Contextual information is considered in some work in order to cope with changes of context or users’ requirements. Choi et al. [17] propose a context-aware workflow system to support dynamically handling changes of services or context at runtime. The suggested workflow system uses uWDL as
a workflow language, in which a context is described as an RDF-based triplet. Contextual changes can be handled at runtime by modifying the workflow definition for a specific workflow instance. Jarouchech, Liu, and Smith [31] present an aspect-oriented framework for generating process variants that correspond to contextual changes based on original processes. The proposed framework comprises a process model, a context model, an evolution model, and a linkage model. Evolution fragments and evolution primitives are introduced in the evolution model to capture changes. The adjustment of a particular process can be realized at both the process schema level and the process instance level.

QoS changes Adaptability of service-based processes to non-functional requirements such as QoS properties is studied in [29,30,57,84]. In order to optimize processes when QoS changes, Harney and Doshi [29] propose a method called value of changed information (VOC) to calculate the impact on Web service-based business processes. In [30], service expiration times obtained from service level agreements are used to reduce the computational overhead of VOC. Wu and Doshi treat the problem of adapting Web service-based processes as a decision-making problem. Service managers are defined for accomplishing activities in a BPEL process. At runtime, service managers will need to communicate with each other based on a regret-based mechanism in order to respond to external changes such as service performance and availability. Agarwal and Jalote [2] present mechanisms for specifying non-functional requirements of BPEL processes as extensions to the BPEL specification. Participant Web services are selected and can be substituted as required at runtime to implement activities of BPEL processes based on the specified non-functional requirements.

Enabling flexibilities of service-based business processes so that they can react quickly and properly to changes especially the changes that are not expected at the design time is a challenging task. As we have seen, various techniques, such as the aspect-oriented programming and the resource description framework (RDF) are used to deal with different types of non-functional requirements that are not addressed by the BPEL standards. The above research suggests extensions to the BPEL specification and mechanisms for capturing and dealing with changes related to non-functional aspects such as security, contextual information, and QoS. In general, to implement BPEL processes with the above extensions or adaptations, existing BPEL engines such as Apache ODE and Active BPEL are extended or integrated with some adaptation engine to enable the various types of process flexibilities. Most of the described approaches have been evaluated using real-world scenarios or examples to demonstrate the feasibility and effectiveness. However, to select a right process specification and the corresponding mechanism for a specific requirement is difficult, due to the fact that there are no common criteria or benchmarks to evaluate the proposed approaches. Moreover, sophisticated tools are also needed to support the management for Web service-based processes such as process evolution.

3.3 Service evolution:

Service evolution refers to the process of continuously modifying a service through a series of consistent changes [56]. We classify the current studies on service evolution into three sub-problems: (i) Web service versioning; (ii) service protocol evolution; and (iii) service evolution as shown in Table 3. In the sub-problem (i), the focus is on managing different versions of WSDL services, such as extending the current UDDI registry or building a new service registry to support Web service interface change management. In (ii), the emphasis is on the management for evolving service protocols. In (iii), the focus is on providing theorems and mechanisms for managing service evolution abstracting from the current Web service standards.

Web service versioning Efforts [14,22,23,33,34,41] have been made for enabling Web service versioning, that is, allowing Web services to evolve in a disciplined and controlled manner. For example, Kalali et al. [33] put forward the requirements for building a Web service registry, called service-oriented monitoring registry, that can monitor changes of Web services and notify service requestors when the requested Web services are changed. The authors aim to devise mechanisms that are able to track interface changes and the availability of requested Web services. Kaminski et al. [34] propose a service design technique, called chain of adapters. For two successive versions of Web services, an adapter that mediates the differences between the new version and the old version is generated. A chain of adapters for a Web service is produced. Each adapter in the chain resolves the incompatibilities of two Web service versions. Thus, the backward compatibility of different versions of a Web service can be achieved. To facilitate Web service versioning control, [14,22] suggest extensions to the current Web service standards. Brown and Ellis [14] advocate the use of version namespace and version numbers in UDDI entry to manage the evolution of Web services. Their approach achieves backward compatibility by allowing multiple versions of a Web service to support the earlier versions of that service. Fang et al. [22] propose a version-aware service model based on extensions to WSDL and UDDI, where WSDL is enhanced to describe the attributes of the service versions, and UDDI is augmented to use versions in a service directory with an event-based notification/subscription mechanism. Differ-ent to the above work, [23] provides an algorithm called VTracker to detect changes in WSDL documents of subsequent versions of real-world services, such as Amazon EC2, PayPal SOAP API3 etc. Based on the detected interface changes, potential effects on the maintainability of service systems are analyzed.
Service protocol evolution:
Evolution of service protocols in Web service environments is also studied by a few efforts. This branch of research is similar to the work done on workflow process evolution, in which change management for workflow schemas and instances migration is extensively studied. In the context of service protocols, the problem of evolution is discussed from a static aspect (modifying protocol definitions) and a dynamic aspect (managing running protocol instances when the corresponding protocol definitions change especially for long conversations). Ryun et al. focus on dynamic protocol evolution where ongoing conversations (protocol instances) need to be handled properly when the protocols change. The authors also address the issue of active conversation migration when there is no formal protocol for this conversation. A set of methods to analyze the impact of a protocol change on the active conversation are devised. The forward compatibility and backward compatibility are defined as a foundation to handle the migration of protocol instances when their protocol schemas have evolved. Then, how to migrate a protocol instance can be determined based on the impact analysis results. Skogsrud et al. focus evolution of trust negotiation protocols. Types of violations are identified when a trust negotiation protocol is changed. Protocol management operators for handling ongoing trust negotiations during a protocol change are developed. Service evolution Theorems and guidelines for service evolution management that abstract from current Web service standards are proposed in [5,6,56,]. Papazoglou classifies service changes into shallow and deep changes according to the effect and side effect they cause on the entire process value-chain. A theoretic approach is introduced for handling shallow changes such as structural changes and protocol changes. To manage deep changes such as behavioral changes, a change-oriented service life cycle methodology is proposed. The major phases in this life cycle include as follows: the initial phase: understanding the causes for a specific service change and determining the change scopes; the second phase: providing in-depth understanding of the service change; the third phase and fourth phase involve service alignment and implementation. Andríkopoulos et al. [5] provide an approach for managing service evolution based on a service specification reference model that is independent of current Web service technologies and captures the main characteristics of different service description models and technologies. Evolution of a specific service is realized by applying a series of change operations on its service elements. This service specification reference model provides a formal foundation for defining the concept of consistency of service schema evolution and conformance of service schema versions. In [6], Andríkopoulos et al. define the concept of service contract to achieve controlled service evolution. The ultimate goal is to enable independent evolution of services and at the same time preserves the interoperability between two interacting services. A service contract specifies the provided and required functionalities of the interacting services. Interoperability between interacting services is then discussed based on the notion of service contract. The concepts of contractual invariance and contract evolution are defined to handle changes occurred in interoperating services.

Although theorems, guidelines, and mechanisms are suggested to enable service evolution management, approaches and techniques for managing the evolution through the entire service life cycle are still lacking. Also, implementation and evaluation of the reported results in the literature are inadequate. Evaluations of these approaches are illustrated by examples or some real-world Web services. Comparison between approaches is difficult due to the lack of common criteria and benchmarks. Also, scalability of these mechanisms is not addressed.

Change analysis and management in service compositions:
In this subsection, we shall discuss the research on change analysis and management for service-based applications and systems. In the SOC paradigm, business processes can be wrapped and exposed as services and complex business processes can be composed using services from different providers. In this context, changes in services and business processes will affect each other as a result of the dependency relations between them. The focus of this section of research is on change detection, impact analysis, and management in service-based applications. In particular, in the context of business collaboration, where service-based business processes are normally cross-organizational boundaries, changes that happen to services and business processes in one party may have various types and levels of impact on the other party. The Bouguettaya group [3,4,42,43] investigates the change problems in the so-called virtual enterprises (VEs) and service-oriented enterprises (SOEs). A VE or a SOE is a service composition that is formed to achieve a business goal. In [4], Akram, Medjahed, and Bouguettaya focus on the problem of ensuring service requests in a dynamic Web service environment. The changes of Web services concerning service request are categorized into two types: the internal change (the change about the information provided by a Web service) and the external change (the existence change of service operations and the service itself). The proposed architecture is based on two key ideas: using ontology to organize Web services and agents to manage changes. In [3], the change problem is addressed in the context of semantic web. The changes occurring in VEs are categorized into three layers: business, ontological, and service. Changes at the business layer are further classified into three types as efficiency change, regulatory change, and development change. Changes at business layer are mapped to the ontological layer that comprises a collection of Web services organized based on ontologies. Changes at the ontological layer are mapped onto
the service layer. The approach involves three steps as detecting changes, propagating changes, and reacting to the changes. Also with the help of ontology, Liu et al. [43] focus on designing change reactions. They propose the concept of Web service ontology to support the tasks of modifying service composition schemas and selecting candidate Web services. A node in a Web service ontology is a service concept that defines a type of Web services within a specific domain. Two types of relationships: inheritance and dependency are used to connect service concepts in a Web service ontology. In [60], Xumin Liu et al, proposed a framework for efficiently managing top–down changes in LCSs, focusing on changes that result in the replacement or addition of a Web service. We proposed a two-phase optimization approach where in the first phase, the selection of Web services is based on using reputation as the key parameter. In the second phase, the non-functional QoWS is used to narrow down the set to those Web services that are both reputable and best meet the QoWS.

Change problems in the context of a cross-organizational setting are studied in [61,62]. Each trading partner has its private business processes, and the corresponding public views exposed for its partners. The change problem to be solved is as follows: if the public view of a partner changes (this change may originated from the associated private process of this partner), how this change will affect the public views of the related partners and in turn their private processes. In [82], a private process is described by BPEL and further converted into a nested word automata (NWA) [83]. A public view for a particular partner is derived from the corresponding private process, which is described by a finite state automata. Two basic types of changes: subtractive changes and additive changes are considered. A subtractive change describes removing a message sequence while an additive change is to add a message sequence in the corresponding public view. When such a change happens in a public view, it is automatically propagated to the corresponding private process.

Similarly, Rendered et al. aim to solve the problem: if a private process of a trading partner changes, how it will affect the public process of the partner and the private and public processes of other interacting trading partners. In other words, when there is a change in a process of a choreographies, what is the impact on the processes of other partners involved in this conversation. Choreographies between trading partners are built on public views which describe how this private process exchanges messages with a particular partner. They also consider the two types of basic changes: adding a message sequence and deleting a message sequence. When a change occurs in a private process, the public views of the private process are generated. Then, this change will be propagated to the public views of its partners to assist the automatic adaptation of public processes.

Ryun et al. focus on managing changes of business protocols in Web service environments. The authors provide a set of change operators for supporting modifying protocols. In addition, they present mechanisms for analyzing change impact of business protocols on service compositions.

Mechanisms for managing running protocol instances when the corresponding business protocols are changed are devised based on the change impact analysis. Skogsrud et al. focus on the change management of security protocols in Web services environments. In particular, they emphasize on changes of trust negotiation protocols. They provide a framework to support the change impact analysis of trust negotiation protocols on the ongoing trust negotiations automatically.

Currently, most of the work evaluates the proposed mechanisms by some real-world examples or scenarios that are different to other research, which makes the comparison difficult. We believe an overall evaluation approach and a test bed are necessary. Moreover, in addition to the above efforts, the following issues relating to Change Analysis and Management in Service Compositions still remain to be solved as follows:

**Modeling techniques for complex service-based systems:**

Modeling techniques are still lacking for complex service-based systems, where services and business processes have complicated relationships. Complex business processes can be wrapped and exposed to partners as multiple business services. The dependencies between services and its under-lying business process need to be identified and formally defined as the foundations for change analysis and management. In [11], the concept of composition tree is proposed for characterizing dependency relations between services in a service composition and SLAs when analyzing the impact factor of a component service. Still in change impact analysis, a typical type of dependencies between services and business processes, where multiple services are associated with one underlying business process, is captured in . Thus, if one service changes, how it impacts on its underlying business process and the associated services can be analyzed. To fulfill the challenging tasks of change management and evolution of service-based systems, these efforts are insufficient for change analysis in complicated cross-organizational business collaboration involving multiple services and business processes; modeling techniques are required for describing dependencies between elements (e.g., messages, operations, activities).

**Support for change detection, analysis, and reaction:**

The above-described studies present some prototype tools and components for the change detection, analysis, and reaction tasks in a service-based system. However, more sophisticated mechanisms and tools support for the change management tasks, including automatic change identification, impact analysis, and
change reaction strategy selection in long-term complex business collaboration are required. A specific change happening to one party may have different level of impact and ripple effect on other collaborating parities’ services and business processes. Thus, it is necessary to understand what types of changes may occur, calculate the exact impact scope, and determine the types of impact before change reactions can be taken. Wildish et al. address the issue of change propagation between business processes that are described by different but related process models within one organization. Changes occurring in a source process are propagated to a related process. The concept of a change region is proposed for process analyst to handle change propagation. In change impact analysis has been studied from the perspective of an organization itself, that is, how changes may impact on its supporting business process. The main idea is to define a set of change impact patterns for capturing different types of change effects. As shown above, [65] have studied the change issues in the context of cross-organizational context. However, only two basic types of changes are examined. More efforts are still required for change impact analysis and management in more complicated cases, that is, various types of changes and complex dependencies between elements in service-based systems.

Non-functional change management:

The above work mainly concentrates on functional changes in services. The analysis and management for non-functional changes including QoS and policy changes are not fully investigated. Very limited efforts have been made for dealing with QoS changes in service compositions. Liu and Bouguettaya [42,43] have suggested approaches for handling QoS changes in service-oriented enterprises. In [11], changes impact of QoS values on the overall service composition is studied. Impact factor of a component service is calculated by considering the dependencies in composition structure and SLAs, to measure the importance of a Web service in a service composition at runtime. A branch of research relating to this issue is the process flexibility, which we have reviewed in the previous section. However, these results are still insufficient to understand the impact of non-functional changes in service-based systems and react to them so that these changes are made transparent to end users. For instance, a change of critical QoS properties may have deep impact on the entire system. Also, management support is lacking for the higher level of changes in service systems such as policy changes. As a result, sophisticated mechanisms and tools support are demanded for analyzing and reacting to these non-functional changes in business collaborations

Conclusion:

This paper provides an extensive review of the current research on change management in the domain of service-oriented computing. Firstly, we have classified the major types of changes that may happen to services into three broad categories: interface change, behavioral change and non-functional change. Then, we have reviewed the main research on managing these types of changes from four broad categories: service adaption, process flexibility, service evolution and change analysis, and management in service compositions. In each sector of research, we have discussed the major approaches proposed in the literature to coping with various types of change problems and pointed out critical issues that remain to be solved. The results obtained from these researches provide guidelines, approaches, and techniques for achieving change automation and making changes transparent to end users. More efforts are required for managing non-functional changes including QoS changes and higher level of change: policy-induced changes in the context of cross-organization collaboration.

REFERENCES


23rd international conference advanced information, systems engineering, pp. 48-61.