Towards Service Co-evolution in SOA Environments: A Survey

Huu Tam Tran[1(✉)], Van Thao Nguyen[1], and Cong Vinh Phan[2]

[1] Distributed Systems Group, Kassel University, Kassel, Germany
{tamth,thaonv}@uni-kassel.de
[2] Nguyen Tat Thanh University, Ho Chi Minh, Vietnam
pcvinh@ntt.edu.vn

Abstract. In a Service-Oriented Architecture (SOA), the need for service evolution comes from service providers and their clients due to changes in requirements and environments. However, enabling controlled service evolution is a critical challenge for the developers since services may be part of different business processes and depend on other services. This paper presents the state of the art of service evolution and in particular, of service co-evolution. The paper also gives an outlook to an emerging trend named microservice and analyzes its advantages and challenges related to service co-evolution. The survey aims to provide a technical overview document to researchers and practitioners who are building industrial-strength adaptive applications related to service co-evolution.

Keywords: Service co-evolution · Service evolution · Service-Oriented Architecture · Software evolution · Service adaptation · SOA

1 Introduction

Service-Oriented Architecture, or SOA, has become a widespread paradigm that provides a flexible IT infrastructure to cope with the increasing pace of business changes and global competition for more than 20 years. As time goes by, our private life and business have been increasingly dependent on SOA applications [1]. Besides, the appearance of Cloud Computing [2] and the Internet of Things [3] or the Web of Things [4] has reinforced this trend even more. The downside of this trend is increasing the complexity of service landscapes. This complexity is due to the sheer size of these systems, the interdependent services, the different levels of service abstractions, the microservice environments and the like. This complexity makes service management as a whole a very substantial challenge for service providers [1].

The fundamental building block of SOA is a service that may play any of the three roles involving service provider, service broker and service consumer. An SOA service is referred to as a discrete entity that can be accessed remotely,

P. C. Vinh and A. Rakib (Eds.): ICCASA 2020/ICTCC 2020, LNICST 343, pp. 233–254, 2021.
https://doi.org/10.1007/978-3-030-67101-3_19

acted on, and updated independently. In the SOA context, the term service evolution is defined as a continuous development of service and expressed by the provisioning and decommissioning of different variants of the service called versions [35]. Another term named service co-evolution refers to a coordinated service evolution [1,10]. Up to date, both of these terms are critical ingredients of the service life-cycle.

A critical question related to SOA services is why we need service evolution and coordinated service evolution (hereby, we denote these terms as (co) evolution). On the one hand, a service (co) evolution is raised in order to satisfying requirements from clients [7]. The requests for adjustments concerning services are usually started from clients for additional functionalities or bug reports. And if the requirements are not satisfied, the clients may switch to another offer. On the other hand, service providers adapt and evolve to the new market trends and attract more clients. Thus, the need for a rapid (co) evolution comes from both service providers and their clients due to the change of requirements and environments [9].

Furthermore, in distributed networks of large-scale environments, every service may depend on other services to avail of certain functionalities. It implies that a change in one service can result in changes in the other service. To prevent outages and failures caused by individual service modifications and updates, coordinated changes are required in such complex systems, i.e., interdependent services must evolve together to maintain compatibility and ties. This phenomenon is called the co-evolution of involved services and contains the meaning of coordination and cooperation of services for co-evolving activities of changes.

Service co-evolution can be seen as a particular case of service evolution [10]. Currently, there is limited support for software service co-evolution since the issue of co-evolution presents intrinsic difficulties. Therefore, this study is intended to systematically review the available literature on service evolution towards service co-evolution.

Our main contributions are as follows: first, we explore and develop a taxonomy for evolutionary changes in SOA environments; second, we review existing approaches for service (co) evolution; third, we discuss and analyze challenging issues related to service (co) evolution.

The rest of the paper is structured as follows: Sect. 2 presents briefly the foundation of service evolution, beginning from software evolution. Section 3 proposes a change taxonomy. Section 4 analyses existing solutions supporting service (co) evolution. Section 5 identifies particular challenges regarding microservice environments. Finally, Sect. 6 concludes the paper.

2 Foundation

In this section, we explore the related contributions in the field of service (co) evolution. We firstly introduce the early contributions to software evolution as the fundamentals of service evolution. In this area, the majority is focused on the laws, the model of software or architectures to facilitate software evolution.

Over time, software evolution has been facing various challenges in order to adapt to new requirements from service providers or service users. In this stage, researchers tried hard to develop the architectures, tools and frameworks to enable the software to be reconfigured at runtime on-the-fly.

2.1 Software Evolution

Software evolution is of great importance in satisfying user requirements under specific changes in the environment [9]. Substantial works have been conducted in related areas promoting software evolution.

In the early days, the term software evolution referred to general software maintenance and configuration. Later, this term has been distinguished as a phase that adapts application software to ever-changing user requirements and operating environments. At the same time, the term maintenance is used to refer to post-delivery activities [11,38].

Nowadays, it is widely accepted that continuous changes are a critical feature of evolution. One of the important research works on software evolution was investigated by Lehman and his colleagues [12], who presented the famous eight laws of software evolution. On the one hand, these laws describe a balance between forces driving new developments. On the other hand, the laws force that slow down progress. In their studies, Lehman et al. [12,13] considered that the changing and adapting requirements from the real-world software systems drive the application to evolve with inevitable and continual feedback. The authors concluded that evolution is an intrinsic and feedback-driven property of software.

Even though Lehman's laws of software evolution have been widely accepted and became basic knowledge of software engineers, there has been no transparency systematic work to fully validate the laws on software quality evolution [14]. Furthermore, most of the rules are just for solving the problems in general static maintenance or software evolution.

In practice, there is a large body of research results available for managing software evolution. These studies fall into the following classifications: (i) analyzing the evolution trend of a software system over a long period of time; (ii) developing effective techniques to support software evolution [38].

At present, researchers have focused on solving two challenging aspects. The first challenge is how to evolve the software. And the second one is how to react with software that has evolved. To the former question, the traditional solution is to improve software development and deployment approaches. With regard to the latter, the users of services have to deal with incompatible interfaces of modified modules and adapt to these changes.

With the advancement of software engineering, the software execution environment has become more dynamic and complex. Traditional methods of maintenance and software evolution face major challenges related to various rapidly changing customer requirements. Therefore, some researchers have proposed new models and tools for evolving the software at runtime. For instance, DYnamic MOdification System (*DYMOS*) by Insup Lee [15] presented general principles for modifying the running software system. Later, *K-Component* by Dowling et

al. [16] defined a meta-model for software architecture to provide possibilities for dynamic adaptation. In 2003, *OSGi* (Open Service Gateway initiative) was proposed by researchers at IBM and SUN aiming to improve the practical use of the limited resources in the embedded devices. These architectures are important in the field of software evolution [17].

In general speaking, researchers described these solutions for software evolution as *static software evolution* and *dynamic software evolution*. The static software evolution refers to refactoring the software at the development stage and then install the new version by shutting down the running application. The static software evolution could make the software temporarily unavailable and thus it may cause delays for enterprise business. The dynamic evolution refers to adjusting the behavior of the software at runtime without breaking down the business. Clearly, dynamic evolution improves the software adaptability and can be more competitive in the modern complex and distributed environments. However, the dynamic software evolution also poses more complex challenges as users attempt to apply themselves to the large-scale distributed environment.

Besides, through an extensive survey, Feng-lin et al. [18] discovered that software evolution is closely related to the changing code, module and architecture and its most recent work is on evolution requirements.

In summary, the existing approaches have provided a strong foundation in the development of software evolution. However, these approaches lack aspects of coordinated, distributed evolution where the resolution of service dependencies have to be negotiated with other services.

2.2 Service Evolution

Let us first briefly define the basic terminology used in this paper. This is necessary because there is no general agreement on the terminology in the wider service (co) evolution community.

2.3 Definitions

A service in SOA is defined as a software component that provides certain capabilities over a network to service consumers in a loosely coupled fashion. A service is encapsulated and offers a clearly defined service interface to service consumers. Services may have different owners with different business agendas and they are free to evolve in an independent fashion [5].

According to Papazoglou et al. [5,7], **service evolution** is *"a continuous process of service development through a series of consistent and unambiguous changes"*. It is expressed through a service's different versions and the key challenge is the forward compatibility between different versions. Similarly, a definition of service evolution is given by Wang et al. [19] who stated that service evolution is *"the process of maintaining and evolving services to carter to new requirements and technological changes"*. Both of these definitions have the same views that service evolution involves the deployment of a new service version, which is caused by necessary changes in interface structure, functionality, usage

protocol, usage policies, business rules and regulations and more. Service evolution can be viewed on two levels of abstraction, which are linked to the macro and micro perspective. From a macro perspective, services in a network are like individual organisms of a biological population. From a micro perspective, interfaces and interaction protocols up to the service are similar to the physical and behavioral characteristics of an organism. So far, many techniques and tactics have been used to ensure customer compatibility with the service.

In distinguished studies on service evolution [5,7], Papazoglou et al. provided the fundamentals for service evolution until the present, especially when they distinguished two kinds of changes named **shallow changes** and **deep changes** based on the nature of service evolution.

- **Shallow changes** are "small-scale, incremental changes that are localized to a service and/or are restricted to the consumers of that service", for shallow changes, typically it could lead to mismatches between services at two levels: (i) interface level (i.e., structural), (ii) interaction protocol level (i.e., behavior, such as messaging order mismatches). To solve the shallow changes, the authors proposed a structured approach based on a robust versioning strategy to support multiple versions.
- **Deep changes** are "large-scale, transformational changes cascading beyond the consumers of a service possibly to consumers of an entire end-to-end service chain." Deep changes include operational behaviour changes and policy-induced modifications. Deep changes rely on the assistance of a change-oriented service life cycle methodology to respond appropriately to changes [35]. To date, deep changes are a challenging and open research problem in the context of service engineering.

Furthermore, Papazoglou [5,7] defines three following concepts:

(i) **Version compatibility** means when we can introduce a new version of either a provider or client of service without changing the other.
(ii) **Backward compatibility** means when a new version of a client is introduced to the providers are unaffected. The client may introduce new features but should still be able to support all the old ones.
(iii) **Forward compatibility** refers to the old version of a client application that could interpret new operation(s) or message(s) introduced by a service.

Some types of changes that are both backward- and forwards-compatible involve the addition of new service operations to an existing service definition, the addition of new schema elements within a service that are not contained within previously existing types [7].

The key problem of service evolution is that the compatibility between the service and its consumers may change when the service evolves. One of the major objectives of the research on service evolution is to reduce the unexpected effects caused by incompatibilities.

We adjust these definitions of service evolution in [5,7,19] for service co-evolution by giving a simple notion, i.e., **service co-evolution** stands for a

coherent process of evolving and maintaining service and its interdependent services through a series of explicit changes. We further explain this domain by considering the dependency graph shown in Fig. 1 of different service providers offering various services. It shows the service dependency graph S which is a set of different services.

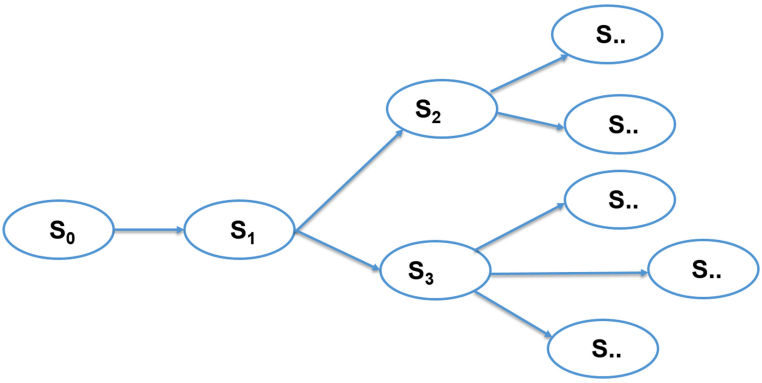

Fig. 1. Dependency graph in service co-evolution scenario

Now, consider node S_0, which is actively connected to all other nodes providing different services without any interruptions in the edges. Similarly, S_1 is connected to the nodes S_2 and S_3 and these nodes are further connected to various other nodes. Considering a scenario, in which S_2 evolves to a new version but S_3 is not affected at all proves a successful service evolution. In this case, the change is confined to the clients of S_2 only. However, in case the evolution of S_2 implies that the nodes S_0, S_1 and S_3 should also evolve in a coordinated fashion, we call this as a co-evolution of service where the evolution is required to update the interdependent services.

On the topic of evolution, we cannot fail to mention adaptation, the counterpart of it. **Adaptation** is a crucial mechanism to keep software and service behaving as expected under certain conditions [5,7]. In service engineering, **adaptation** usually refers to select candidate services when suffering from failure, (re-) composite services for changed business process and distribute new service instances in adapting to emergent workload [7]. The principal difference between adaptation and evolution is that adaptation will not change the service itself, but evolution will [5].

3 Proposed Change Taxonomy

Different change taxonomies for service evolution have been developed over the years. In general, various change taxonomies proposed base on the effects or scope of changes during the interaction process.

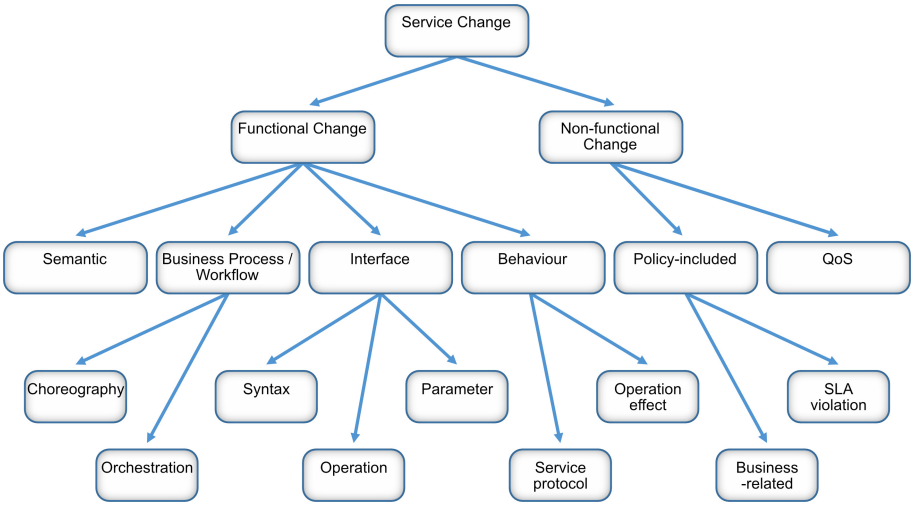

Fig. 2. Proposed change taxonomy

According to Treiber [36], these evolutionary changes involve changes in requirements, changes of the service interface, changes in implementation and QoS [36]. These change activities come from different requirements such as developers, providers, users and service integrators that interact in the interdependent services.

To date, evolutionary changes in existing research works can be defined into various categories such as change of interface, change of semantics protocols, change of requirements and change of business process models.

In this survey, we focus on the solutions of "not compatible and resolvable changes" [1] such as interface, semantic, business process model (workflow) and behavior changes. We distinguished them as follows:

- *Interface changes* refer to modify the signature of the service interface.
- *Semantic changes* refer to modify service properties as declared in interface annotations.
- *Behavior changes* refer to any transformation or modification of service behavior.
- *Business process/workflow changes* reflect a modified business process model of service composition. The demand for changing business processes (internal organizational level) arises due to various reasons, such as new regulations or the emergence of new competitors at the market.

These changes are described in detail in Table 1. We adopt a change taxonomy shown in Fig. 2. The proposed taxonomy is an extension of the classification of evolutionary changes in [23]. Our terminology is consistent with the one used in [21] and shown in Table 1. Some other kinds of changes such as QoS, policy, parameters, optional operations are defined similarly to the previous works in [20,21].

Table 1. Change terminology

Category	Type of change	Characteristic
Non-functional	QoS	– performance of service properties e.g., server load, concurrent users – performance of network latency, – performance of throughput
	Policy	– change in policy assertions on services, which specify business agreements
Functional	Behavior	– change in the service protocol i.e., prescribed invocation, operational behavior
	Semantic	– cover all changes that are not involved description of services, operations, parameters or return values
	Interface	– change in the interface signature e.g., parameters, operations, message structure – addition of new functionality or the update of existing functionality – interface changes may affect the implementation, QoS, pre-condition, post-condition, usage of the service
	Business Process/ Workflow	– modify the business process model – change in choreography or orchestration model of service composition

4 Positioning Approaches

Existing works for supporting service (co) evolution have mainly focused on the following steps: change detection, change impact analysis and reaction. Their sequences of these processes are depicted in Fig. 3:

(i) Change Detection (CD). Change detection is a critical process in service evolution management. It helps affected services to find out of changes as well as kinds of changes that can be used as the input data for analyzing the impact levels. Researchers classified evolutionary changes into different types of critical changes. However, in many cases, changes in behavioral usually are more complicated and need more effort to adapt by considering the actual values communicated between services and their clients.

These approaches (see Table 2) mainly focused on service interfaces, workflow, semantic and behavior change. These approaches have provided many tools or frameworks such as: *Vtracker* [24], *WSDarwin* [25], WSDLiff [26], *DiCORE* [27]. During the time of this survey, we realized that there are a few works considering behavioral changes.

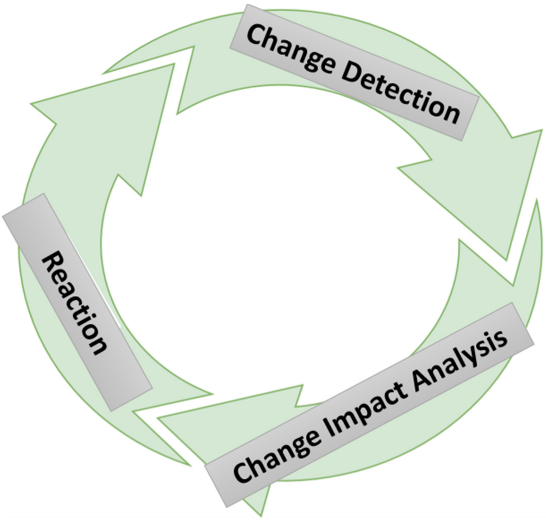

Fig. 3. Service evolution development cycle

(ii) Change Impact Analysis (CIA). The goal of this process is to understand the relationship between the service and the change. The service users should know which parts of the system will be affected by the change and examine them for additional impacts since the modification in one part of service may have subsequent effects on other related services. Significant literature works in this area can be classified into two categories techniques: dependency analysis and trace-ability analysis [28]. Through an impact assessment, it is possible to evaluate the change effects and procedure evolution strategies to reduce risks and maintenance costs.

The output of existing approaches for this phase, usually are models, techniques or design patterns, such as Trust Dependency Graph [29], Versioning Model [30], Dependency Model [31], Change Pattern [19], DiCORE-CIA [27]. Furthermore, we found that there has been special attention paid to the topic of change impact analysis from the research community.

(iii) Change Reaction (CR). This process may involve other steps such as decision-making, propagation of changes and an optional broader changes context to support other affected services (in case of coordinated co-evolve services), eventually giving a set of prioritized actions to adapt new changes. Propagation of changes addresses how the impact of a change can be effectively propagated to other entities with minimal ripple, or what additional changes are required for a service to maintain consistency. As new functionality is added or changed in services, developers must ensure that other system entities are updated and consistent in response.

Existing approaches have mainly focused on changes in business process models (workflow). Some important frameworks are DYCHOR [32] and C3Editor

[33]. The DYCHOR framework enables evaluating the change propagation of a process in choreography on the basis of an extended automated model, while the C3Editor visualizes the different models and enables the definition and application of changes to the private models. Besides, the C3Editor determines and visualizes the partners affected by a change and the updates required for change propagation.

In our view, these approaches could support effectively the coordination of change processes. Additionally, we suppose that changes in a business process model usually require manual intervention by developers. Thus, such changes are out of the scope of automated service (co) evolution support and not further considered in this survey.

4.1 Support Service Evolution

In the scope of evolution services, researchers have spent significant effort on investigating methods and techniques for the management of service changes. These researches can be seen as a precursor and the fundamental for research on service co-evolution since service co-evolution is a special case of service evolution.

Table 2 lists important approaches related to service evolution aspects such as authors, key contributions, available software (denote Yes (Y) or No (N)), and support processes (e.g., change detection (CD), change impact analysis (CIA) or change reaction (CR)). In our view, all of these approaches have high-value results in the field of service (co) evolution in recent years.

These approaches could fall into one of the following categories: (I) Tool/Model-based, (II) Versioning-based, (III) Pattern/Adaptor-based and (IV) Analysis of Change Impact-based. Naturally, some approaches might belong to more than one category, for instance, the work by Khebizi et al. [34] provided a framework, a software tool and patterns to support dynamic change management of business protocols.

(I) **Tool/Model-Based**
One of the first works handling the problem of service evolution is developed by Treiber et al. [23]. Their work addressed two main problems related to services: (a) what type of information is required for a particular perspective, how all types of information are integrated into a single model, and (b) how these types of information are managed. The first problem concerns the development of an aggregated, flexible and extensible information model for services. The latter is linked to the development of a management framework that can track historical information.

Romano et al. [26] proposed the WSDLDiff tool that can be used to derive the set of delta changes applied to a service. This tool considers the syntax of the WSDL file and the schema file XSD that is used to define the data types of the WSDL interface. Similarly, Li et al. [46], presented critical empirical studies about the most common types of service changes.

Another important framework comes from V. Andrikopoulos et al. [21], who introduced a service specification reference model and introduced the concept of service evolution management. Based on the type and set theory as well as the service specification model, the authors developed an approach to reason and identified the conditions (i.e., a set of changes) under which services can evolve while preserving compatibility. However, their work only focuses on the preventive evolution model and do not pay attention to the impact and adaptation aspects [38].

Fokaefs et al. [24] also published empirical results of evolution scenarios and presented the Vtracker. Specifically, the authors created an intermediate XML representation to reduce the verbosity of the WSDL specification. However, Vtracker does not take into account the syntax of WSDL interfaces. An upgraded Vtracker named WSDarwin [25,40], which can be used to automatically identify changes between different versions of service by comparing interface description documents. WSDarwin tool provided a solution to answer the question of how a client application can be supported in adapting to changed services. However, WSDarwin does not indicate how the Web Service provider could perform the adaptation assistance or how to deal with the dependencies when generating and compiling the client stub.

Latter, Zou et al. [38] proposed a change-centric model in which necessary changes are identified, planned, implemented, tested and then notified to all necessary stakeholders. In the model, the delta is a set of changes from one version to its next version of the service.

Recently, Jahl et al. [27] developed the DiCORE framework that determines the kind of changes, categorizes the changes and shares them with dependent clients. The framework helps the developers to find out the structural changes in workflow. In summary, these tools and models provided practical ways to address the evolutionary challenges, such as detecting changes and analyzing the change impact, eventually supporting related services during an evolution process.

(II) **Versioning-Based**

Versioning is a traditional and practical way to address the incompatibility issue [18,45]. A robust versioning strategy allows for service upgrades and improvements while continuously supporting previous versions. Leitner et al. [20] presented a comprehensive versioning approach specifically for handling compatibility issues, based on a service version graph and version selection strategies. The proposed framework is used to dynamically and transparently invoke different versions of service through service proxies. In this path, Kaminski et al. [22] outlined various requirements for versioning and demonstrated why common versioning strategies are inappropriate in the context of services. The authors proposed the Chain of Adapters pattern [22] for developing evolving services. However, the adapter does not support the parallel execution of different service implementations. Also, adapter implementations may be faulty and break old clients [45].

In order to fix this faulty and do not break old clients, Weinreich et al. [45] proposed a versioning model for supporting the evolution of service-oriented

architectures. The model involves a set of services into a subsystem and assigns them the same version identifier. Even if only one service is changed, all services within the same subsystem will be tagged with a new version number. Consequently, multiple versions of the same subsystem may co-exist. Becker et al. [56] proposed an approach to automatically determine compatibility that could be applied with the compatibility pattern. Similarly, Yamashita et al. [30] introduced a novel feature-based versioning approach for assessing service compatibility and proposed a different versioning strategy, following the W3C standards.

In fact, various versioning approaches are proposed to address the challenges of the service version. At the technical level, these approaches relied heavily on the SOA [18]. In general, they are used together with the design pattern and related tools that will be presented in detail in the next section.

(III) **Pattern/Adaptor-Based**
Design patterns and adapters have been widely used for software development for structuring solutions. For instance, Wang et al. [19] focused on a common evolution scenario in which a single service is provided by a single provider. In particular, the authors proposed four patterns involving compatibility, transition, split-map and merge-map. These patterns provide generic and reusable strategies for service evolution.

It is worth considering the actual work on service compatibility, which aims to assist services consumers in seamlessly transferring their programs to newer versions [56]. Becker et al. [56] proposed an approach to automatically determine compatibility that could be applied with the compatibility pattern.

In case the change is not compatible, the work of Kaminski et al. [22] introduced an adapter-based approach to maintain multiple versions of service simultaneously. The novel idea of this approach is to use a proxy that enables dynamic binding and invocation for client applications to maintain multiple versions of service on the server-side and [41]. At the same time, Frank et al. [47] distinguished between a service interface (public) and its implementation (private).

To address possible interface mismatches, Dumas et al. [48] suggested an algebra over interfaces and a visual language that allows pairs of provided-required interfaces to be linked through algebraic expressions. Benatallah et al. [49] proposed adapters approach using mismatch patterns, which may capture the possible differences between two interaction protocols.

Similarly, H. R. Motahari Nezhad et al. [57] provided semi-automatic support for adapter generation to resolve interface mismatch and deadlock-free interaction incompatibility. Following previous works of Benatallah et al. [49], Ryu et al. [39] studied the protocol compatibility using path coverage algorithms based on finite state machines (FSM) service model and suggested adapter/ad-hoc protocol as solutions. The reason for using FSM service mmodel is because it is a well-known paradigm based on a formalism that is easy to understand for inexperienced users and is suitable for representing reactive behaviors.

In research work [43], the authors tried to keep client applications being synchronized with evolved services through semi-automatic client updates. In their proposed tool, it first analyzes the delta between different versions of service and exports them into a well-formatted document. After that, drawing on the clients' usage history. Next, they employ a consumer code customized component to highlight the client code fragments that need to be updated. In the same path, Ouederni et al. [44] introduced a framework to resolve interface and behavior mismatches by automatically updating the clients based on compatibility measuring. The update process can be parameterized with some user requirements to prevent the behavior that a designer does not want to appear in the client interface.

(IV) **Analysis of Change Impact-Based**
Basu et al. [37] proposed a technique to extract dependencies from log files. The technique could be adapted to infer the amount of dependent service consumers. Once the dependencies are transfarency, it is also essential to infer the impact of service changes on the applications.

Later, Wang et al. [52] proposed his dependency model in analyzing the impact of service evolution. The authors considered the dependency model to analyze the dependency links among services that work in collaborations. This model extracts the degree of dependency for each link between the elements in one service or between services. It is a foundation for most of the later studies in this field. The dependency model proposes a matrix to describe the dependency relations between services and between elements in one service or more services. However, the disadvantages are also obvious: (a) The model assumes that the dependencies are known at design time, which is invalid in the dynamic environment; (b) The model does not distinguish the change types add, remove and modify. It considers that each type of change results in the same impact. Apart from these disadvantages, the dependency model does not explain how and where to obtain the changes, which is important to service evolution.

Yamashita [30] presented an impact analysis based on usage profiles. This method helps service providers estimate the impact on consumers as well as giving an evolution decision later. In the same fashion, Liao et al. [29] proposed a trust dependency graph that is introduced to analyze the impact on the trust of component services.

Latterly, Jahl et al. [27] also provided framework DiCORE to analyze the impact of changes based on the patterns. This framework allows an intuitive and graphical formulation of patterns while other existing tools completely ignore user-defined change patterns.

Table 2. Existing approaches support service evolution

ID	Author/Reference	Key Contribution	Focus on /Available Software	Process
1	Papazoglou and Andrikopoulos [6,35]	Classify and analyze shallow and deep changes A set of theories and models that unify different aspects of services (description, versioning and compatibility)	All kind of changes/ N	CD, CIA, CR
2	Treibe et al. [36]	SEMF framework to manage the changes on interface, interaction patterns or QoS aspects	Requirement change, Interface change, Implementation change and QoS change/N	CD, CR
3	Romano et al. [26]	A tool to extract changes automatically	WSDL document change/Y	CD
4	Forkaefs et al. [24]	Vtracker tool identifies changes between different version of a service	WSDL Interface change/N	CD
5	Forkaefs et al. [25,40]	WSDarwin tool supports the clients to coevolve with the provider service	WSDL, WADL interface change/Y	CD
6	Wang et al. [19]	A service evolution model A method to analyze service dependencies Proposed four patterns: compatibility, transition, split-map, merge-map	Operation and data type change/N	CIA
7	Basu et al. [37]	A tool can extract dependencies from log files.	Protocol change/Y	CIA
8	Kaminski et al. [22]	A chain of adapters technique approach for deploying multiple service versions	WSDL Interface change/N	CD
9	Zou et al. [38]	A change-centric model and a method for the change impact analysis	WSDL interface change/N	CD, CIA
10	Ryu et al. [39]	A framework supports business protocol	Protocol change/N	CD, CIA
11	Leiner et al. [20]	End-to-end versioning support based on service history	Interface change/N	CD
12	Frank et al. [47]	A service interface proxy for dealing with the incompatibility	WSDL Interface change/N	CD
13	Dumas et al. [48]	An interface adaptation method, in which each interface is represented as an algebra expression that could be transformed and linked accordingly	Interface change/N	
14	Benatallah et al. [49]	Adapters as an approach based on mismatch patterns which captures the possible differences between two interaction protocols	Protocol change/N	CD

(*continued*)

Table 2. (*continued*)

ID	Author/Reference	Key Contribution	Focus on /Available Software	Process
15	Jahl et al. [27]	Framework DiCORE to determine the kind of changes and the affected components in a business process	Business process change/Y	CD
16	Tran et al. [51, 53]	An approach detects and extracts interface changes	WADL file change/N	CD
17	Wang et al. [31, 52]	A dependency impact analysis the kind of changes and the affected causes and effects of changes.	Operation and data type change/N	CIA
18	Jahl et al. [55]	An architecture captures and assesses the behavior dimension of services	Behavior changes/N	CD
19	Becker et al. [56]	A versioning framework determines compatibility based on patterns	Interface change/N	CD, CIA
20	Motahari et al. [57]	A semi-automatic method for adapter generation for the mismatches at the interface-level, protocol level	Interface change, Protocol change/N	CD, CIA
21	Yamashita et al. [30]	A change impact analysis approach based on usage profile	Operation and data type change/Y	CIA
22	Li et al. [46]	A method for change impact analysis. Proposed 16 API changed patterns	Operation and API changes/N	CIA
23	Weinreich et al. [45]	A versioning model	Implementation change, Interface change/N	CD
24	Khebizi et al. [34]	A framework with migration patterns	Protocol change/N	CD
25	Liao et al. [29]	A trust analysis model	Interface change, Binding change/N	CIA
26	Ouederni et al. [44]	A framework to resolve the compatibility; An interface model	Internal behavior change. Message parameters change/N	CD
27	Le et al. [43]	A framework for facilitating the service consumer; An interface model	WSDL interface /N	CD
28	Olga et al. [42]	An framework for managing semantic, syntactic and protocol changes	Semantic, syntactic, protocol/N	CD
29	Fang et al. [41]	A version-aware service description model; A version-aware service directory model	Implementation and interface change. Binding change/N	CD
30	Thanos et al. [59]	A framework for semantic drift	Semantic change/Y	CD
31	Juric et al. [58]	A model to support versioning interfaces	WSDL interface, Message parameters change/N	CD

4.2 Support Service Co-evolution

The need for advancements in service co-evolution support is undisputed, just as for all software. One of the most challenging aspects of a service co-evolution is the ability to co-evolve services together in order to retain compatibility and bindings. Obviously, support service co-evolution also means support service evolution since it is a particular case of service evolution. In practice, the processes of a service co-evolution require further co-evolve steps, e.g., coordination among inter-dependency parties. Even though the coordinated distributed service evolution plays an essential part in SOA environments, there is currently a lack of attention being paid to this kind of service evolution.

This section highlights some valuable research works that investigated service co-evolution as the following aspects:

- *Classification of evolutionary changes:* One of the first works handling the problem of service co-evolution is developed by M. Papazoglou [35]. The author proposed a fundamental classification of evolutionary changes in the service-based system. Their work also introduces two types of service changes, one of them called deep-changes. In our thesis, we concentrate on coordinated deep-changes in large-scale service computing scenarios. However, M. Papazoglou did not take into account the special scenario of service evolution, where several services that have to co-evolve together in order to retain compatibility.
- *Requirements for service co-evolution:* It is worth to mention this research work by De Sanctis et al. [1] who first propose eight requirements for service co-evolution. However, the authors mainly focus on general requirements. Even though the authors do not analyze in deep of these requirements, but they bring a small step for the developers towards service co-evolution. Furthermore, the authors presented a solution for service co-evolution based on the Domain Object concept, which supports deep-changes across a service dependency graph, through the decentralized collaboration of evolution agents.
- *A distributed knowledge-based evolution model:* Wang et al. [54] proposed a distributed knowledge-based evolution model named *DKEM* to promote competition and collaboration between services from different vendors. The model regards stability as a key factor in competition and a stability evaluation model is intended to calculate the stability of services, vendors and service-based processes. Based on the evaluation model, two evolution patterns are specified with which new and more stable cooperation among services can be examined automatically by utilizing a runtime self-adaption mechanism. The authors reported that the model *DKEM* is effective for competition and cooperation among services with distributed knowledge and evolved processes have higher stability and reaction efficiency. Nonetheless, the model is designed as a solution to eliminating semantics conflicts between different vendors with different ontologies.
- *Process of co-evolution using meta-models in model-driven engineering:* Cicchetti et al. [60], presented atomic changes and defined the process

of co-evolution. The authors also created a differential meta-model with the identified changes. However, in this approach, a number of open questions remain, including issues of systematic validation of the dependency detection, change impact analysis and resolution technique, which necessarily encompasses larger population models and meta-models.

- *A service co-evolution management model and multi-agent architecture for service co-evolution*: Recently, a new research project called PROSECCO[1] [50,51,53,55] provided a general service co-evolution management model and corresponding reference architecture. This architecture involves the different stakeholders, their roles and responsibilities in the service co-evolution as well as the architectural components needed to perform service co-evolution as part of the general service management. The architectural design for service co-evolution equips every service with an agent called Evolution Agent (EVA) that performs the service evolution in collaboration with other agents. The modular EVA architecture makes an EVA easily adaptable for different service environments, such as IoT, web services and microservices.

In summary, during the survey of this work, we found that there are limited results that concentrate on coordinated deep changes in large-scale service computing scenarios. Such evolution support is urgently needed in order to cope with the increasing complexity of SOA environments.

5 Outlook

Recently, a new emerging trend called *microservice* has been one of the fastest-rising trends in the development of enterprise applications and enterprise application landscapes [61]. The term microservice or microservice architecture refers to a new software production model including continuous deployment, continuous delivery and continuous evolving and managing [63].

In a microservice architecture, instead of creating single-layered systems, the development shifted towards a composition of microservices. In that respect, a system is a set of loosely-coupled small services that are isolated in small coherent and autonomous units [62].

Undoubtedly, a microservice architecture is a form of SOA and is considered as the state-of-the-art design concept, which exhibits many advantaged characteristics. One of the greatest advantages of using microservices is the loose coupling, which leads to agile and fast evolution and continuous redeployment [64]. The basic design idea of microservice architectures is to towards a new environment where there are small teams with tight communication between developers and other stakeholders. Each team is responsible for a few small independent services. These teams will work at their own pace deploying new versions of services without considering coordinate services [63]. Consequently, this design is contradictory to service co-evolution requirements that services need coordination of others to achieve their goal.

[1] https://www.uni-kassel.de/eecs/fachgebiete/vs/research/prosecco.html.

To date, microservice architecture has been adopted in different forms by many companies such as Netflix, Google, Apple, Microsoft and Amazon. These companies have recognized that by using microservices architecture, their software product delivery cycles will be shorter. Because there is no barrier in timing to releasing new service's version, thus, their customers may update new versions instantly and transparently [63]. However, a speedy producing amount of service versions may need more effort to handle various versions in a short period of time. Subsequently, this leads to another major challenge for service management and service (co) evolution as well.

In summary, applying microservice architectures brings various benefits for service providers. However, it also has made more challenges to managing service (co) evolution due to the service landscape with an enormous variety of independent services. This complexity makes service co-evolution as a real challenge for developers.

6 Conclusions

This paper presents a survey on frequently cited approaches supporting service (co) evolution in SOA environments in the last decades. The paper aims to provide a technical overview of challenges related to service co-evolution. Furthermore, we propose a new taxonomy of changes based on hierarchy and relationships. Additionally, we distinguish three main processes of service evolution that should be handled during service (co) evolution. Finally, we give an outlook to an emerging trend named *microservice* and analyze its advantages and challenges related to service co-evolution.

Acknowledgements. The work described in this paper was conducted by Huu Tam Tran as a part of his Doctoral study under the supervision of Prof. Dr. Kurt Geihs. The first author would like to acknowledge his supervisor, the Distributed Systems Group - University of Kassel and the DFG project named PROSECCO for the support of his research.

References

1. De Sanctis, M., Geihs, K., Bucchiarone, A., Valetto, G., Marconi, A., Pistore, M.: Distributed service co-evolution based on domain objects. In: Norta, A., Gaaloul, W., Gangadharan, G.R., Dam, H.K. (eds.) ICSOC 2015. LNCS, vol. 9586, pp. 48–63. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-50539-7_5
2. Marinescu, D.C.: Cloud Computing: Theory and Practice. Morgan Kaufmann, Burlington (2017)
3. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of Things (IoT): a vision, architectural elements and future directions. Future Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
4. Pfisterer, D., Römer, K., Bimschas, D., Kleine, O., Mietz, R., Truong, C., et al.: SPITFIRE: toward a semantic web of things. IEEE Commun. Mag. **49**(11), 40–48 (2011)

5. Papazoglou, M.: Web Services and SOA: Principles and Technology, vol. 2. Pearson Education Limited, Harlow, Essex (2012)
6. Andrikopoulos, V., Benbernou, S., Papazoglou, M.P.: On the evolution of services. IEEE Trans. Softw. Eng. **38**(3), 609–628 (2012)
7. Papazoglou, M.P., Andrikopoulos, V., Benbernou, S.: Managing evolving services. IEEE Softw. **28**(3), 49–55 (2011)
8. Ouederni, M., Salaün, G., Pimentel, E.: Client update: a solution for service evolution. In: IEEE International Conference on Services Computing (SCC), pp. 394–401 (2011)
9. Naji, H., Mikki, M.: A Survey of Service Oriented Architecture Systems Maintenance Approaches
10. Tran, H.T., Baraki, H., Geihs, K.: Service Co-evolution in the Internet of Things. EAI Endorsed Trans. Cloud Syst. **1** e5 (2015)
11. Bennett, K.H., Rajlich, V.T.: Software maintenance and evolution: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering ACM, pp. 73–87 (2000)
12. Lehman, M.M., Ramil, J.F.: Software evolution-background, theory, practice. Inf. Process. Lett. **88** 33–44 (2003)
13. Lehman, M.M., Belady, L.A.: Program Evolution: Processes of Software Change. Academic Press Professional Inc., Cambridge (1985)
14. Yu, L., Mishra, A.: An empirical study of Lehman's law on software quality evolution. Int. J. Softw. Inf. **7**, 469–481 (2013)
15. Cook, R.P., Lee, I.: DYMOS: a dynamic modification system. ACM SIGPLAN Notices **18**(8), 201–202 (1983)
16. Dowling, J., Cahill, V.: The K-component architecture meta-model for self-adaptive software. In: Yonezawa, A., Matsuoka, S. (eds.) Reflection 2001. LNCS, vol. 2192, pp. 81–88. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45429-2_6
17. Alliance, O.: OSGi Service Platform, Release 3. IOS Press Inc., Amsterdam (2003)
18. Li, F.L., Liu, L., Mylopoulos, J.: Software service evolution: a requirements perspective. In: IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), pp. 353–358. IEEE (2012)
19. Wang, S., Higashino, W.A., Hayes, M., Capretz, M.A.: Service evolution patterns. In: 2014 IEEE International Conference on Web Services (ICWS), pp. 201–208. IEEE (2014)
20. Leitner, P., Michlmayr, A., Rosenberg, F., Dustdar, S.: End-to-end versioning support for web services. In: IEEE International Conference on Services Computing, SCC 2008, vol. 1, pp. 59–66. IEEE (2008)
21. Andrikopoulos, V., et al.: A theory and model for the evolution of software services. Tilburg University, School of Economics and Management (2010)
22. Kaminski, P., Müller, H., Litoiu, M.: A design for adaptive web service evolution. In: Proceedings of the 2006 International Workshop on Self-Adaptation and Self-Managing Systems, pp. 86–92. ACM (2006)
23. Treiber, M., Truong, H.-L., Dustdar, S.: On analyzing evolutionary changes of web services. In: Feuerlicht, G., Lamersdorf, W. (eds.) ICSOC 2008. LNCS, vol. 5472, pp. 284–297. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-01247-1_29
24. Fokaefs, M., Mikhaiel, R., Tsantalis, N., Stroulia, E., Lau, A.: An empirical study on web service evolution. In: 2011 IEEE International Conference on Web Services (ICWS), pp. 49–56. IEEE (2011)

25. Fokaefs, M., Stroulia, E.: Wsdarwin: studying the evolution of web service systems. In: Bouguettaya, A., Sheng, Q., Daniel, F. (eds.) Advanced Web Services, pp. 199–223. Springer, New York (2014)
26. Romano, D., Pinzger, M.: Analyzing the evolution of web services using fine-grained changes. In: 2012 IEEE 19th International Conference on Web Services (ICWS), pp. 392–399. IEEE (2012)
27. Jahl, A., Baraki, H., Tran, H.T., Kuppili, R., Geihs, K.: Lifting low-level workflow changes through user-defined graph-rule-based patterns. In: Chen, L.Y., Reiser, H.P. (eds.) DAIS 2017. LNCS, vol. 10320, pp. 115–128. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-59665-5_8
28. Dam, H.K., Ghose, A.: Supporting change impact analysis for intelligent agent systems. Sci. Comput. Programm. **78**(9), 1728–1750 (2013)
29. Liao, L., Qi, S., Li, B.: Trust analysis of composite service evolution. In: 2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA), pp. 15–22. IEEE (2016)
30. Yamashita, M., Vollino, B., Becker, K., Galante, R.: Measuring change impact based on usage profiles. In: 2012 IEEE 19th International Conference on Web Services (ICWS), pp. 226–233. IEEE (2012)
31. Wang, S., Capretz, M.A.: Dependency and entropy based impact analysis for service-oriented system evolution. In: Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology, vol. 01. IEEE Computer Society, pp. 412–417 (2011)
32. Song, W., Zhang, G., Zou, Y., Yang, Q., Ma, X.: Towards dynamic evolution of service choreographies. In: 2012 IEEE Asia-Pacific Services Computing Conference (APSCC), pp. 225–232. IEEE (2012)
33. Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with change in process choreographies: design and implementation of propagation algorithms. Inf. Syst. **49**, 1–24 (2015)
34. Khebizi, A., Seridi-Bouchelaghem, H., Benatallah, B., Toumani, F.: A declarative language to support dynamic evolution of web service business protocols. Serv. Oriented Comput. Appl. **11**(2), 163–181 (2017)
35. Papazoglou, M.P.: The challenges of service evolution. In: Bellahsène, Z., Léonard, M. (eds.) CAiSE 2008. LNCS, vol. 5074, pp. 1–15. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69534-9_1
36. Treiber, M., Truong, H.L., Dustdar, S.: Semf-service evolution management framework. In: 34th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2008, pp. 329–336 (2008)
37. Basu, S., Casati, F., Daniel, F.: Toward web service dependency discovery for SOA management. In: IEEE International Conference on Services Computing, SCC 2008, vol. 2, pp. 422–429. IEEE (2008)
38. Zuo, W., et al.: Managing and modeling web service evolution in SOA architecture. PhD thesis, Université de Lyon (2016)
39. Ryu, S.H., Casati, F., Skogsrud, H., Benatallah, B., Saint-Paul, R.: Supporting the dynamic evolution of web service protocols in service-oriented architectures. ACM Trans. Web (TWEB) **2**(2), 13 (2008)
40. Fokaefs, M., Stroulia, E.: Using WADL specifications to develop and maintain REST client applications. In: 2015 IEEE International Conference on Web Services (ICWS), pp. 81–88. IEEE (2015)
41. Fang, R., et al.: A version-aware approach for web service directory. In: IEEE International Conference on Web Services, ICWS 2007, pp. 406–413. IEEE (2007)

42. Groh, O., Baraki, H., Jahl, A., Geihs, K.: COOP - automatiC validatiOn of evOlving microservice comPositions. In: Seminar on Advanced Techniques Tools for Software Evolution, (Posters, Demos), pp. 1–6 (2019)
43. Le Zou, Z., Fang, R., Liu, L., Wang, Q.B., Wang, H.: On synchronizing with web service evolution. In: 2008 IEEE International Conference on Web Services, pp. 329–336. IEEE (2008)
44. Ouederni, M., Salaün, G., Pimentel, E.: Client update: a solution for service evolution. In: 2011 IEEE International Conference on Services Computing (SCC), pp. 394–401. IEEE (2011)
45. Weinreich, R., Ziebermayr, T., Draheim, D.: A versioning model for enterprise services. In: 21st International Conference on Advanced Information Networking and Applications Workshops, AINAW 2007, vol. 2, pp. 570–575. IEEE (2007)
46. Li, J., Xiong, Y., Liu, X., Zhang, L.: How does web service API evolution affect clients? In: 2013 IEEE 20th International Conference on Web Services (ICWS), pp. 300–307. IEEE (2013)
47. Frank, D., Lam, L., Fong, L., Fang, R., Khangaonkar, M.: Using an interface proxy to host versioned web services. In: 2008 IEEE International Conference on Services Computing, pp. 325–332. IEEE (2008)
48. Dumas, M., Spork, M., Wang, K.: Adapt or perish: algebra and visual notation for service interface adaptation. In: Dustdar, S., Fiadeiro, J.L., Sheth, A.P. (eds.) BPM 2006. LNCS, vol. 4102, pp. 65–80. Springer, Heidelberg (2006). https://doi.org/10.1007/11841760_6
49. Benatallah, B., Casati, F., Grigori, D., Nezhad, H.R.M., Toumani, F.: Developing adapters for web services integration. In: Pastor, O., Falcão e Cunha, J. (eds.) CAiSE 2005. LNCS, vol. 3520, pp. 415–429. Springer, Heidelberg (2005). https://doi.org/10.1007/11431855_29
50. Tran, H.T., Baraki, H., Geihs, K.: An approach towards a service co-evolution in the internet of things. In: Giaffreda, R., Vieriu, R.-L., Pasher, E., Bendersky, G., Jara, A.J., Rodrigues, J.J.P.C., Dekel, E., Mandler, B. (eds.) IoT360 2014. LNICST, vol. 150, pp. 273–280. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19656-5_39
51. Tran, H.T., Baraki, H., Kuppili, R., Taherkordi, A., Geihs, K.: A Notification management architecture for service co-evolution in the internet of things. In: 2016 IEEE 10th International Symposium on Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA), pp. 9–15. IEEE (2016)
52. Wang, S., Capretz, M.A.: A dependency impact analysis model for web services evolution. In: IEEE International Conference on Web Services, ICWS 2009, pp. 359–365. IEEE (2009)
53. Tran, H.T., Jahl, A., Geihs, K., Kuppili, R., Nguyen, X.T., Huynh, T.T.B.: DECOM: a framework to support evolution of IoT services. In: Proceedings of the Ninth International Symposium on Information and Communication Technology, pp. 389–396 (2018)
54. Wang, X., Feng, Z., Chen, S., Huang, K.: DKEM: a distributed knowledge based evolution model for service ecosystem. In: 2018 IEEE International Conference on Web Services (ICWS), pp. 1–8. IEEE (2018)
55. Jahl, A., Tran, H.T., Baraki, H., Geihs, K.: WiP: behavior-based service change detection. In: 2018 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 267–269. IEEE (2018)
56. Becker, K., Lopes, A., Milojicic, D.S., Pruyne, J., Singhal, S.: Automatically determining compatibility of evolving services. In: IEEE International Conference on Web Services, ICWS 2008, pp. 161–168. IEEE (2008)

57. Kongdenfha, W., Motahari-Nezhad, H.R., Benatallah, B., Casati, F., Saint-Paul, R.: Mismatch patterns and adaptation aspects: a foundation for rapid development of web service adapters. IEEE Trans. Serv. Comput. **2**(2), 94–107 (2009)
58. Juric, M.B., Sasa, A., Brumen, B., Rozman, I.: WSDL and UDDI extensions for version support in web services. J. Syst. Softw. **82**(8), 1326–1343 (2009)
59. Stavropoulos, T.G., Andreadis, S., Riga, M., Kontopoulos, E., Mitzias, P., Kompatsiaris, I.: A Framework for Measuring Semantic Drift in Ontologies. In: SEMANTiCS (Posters, Demos, SuCCESS) (2016)
60. Cicchetti, A., Di Ruscio, D., Pierantonio, A.: Managing dependent changes in coupled evolution. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 35–51. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02408-5_4
61. Ghofrani, J., Lübke, D.: Challenges of microservices architecture: a survey on the state of the practice. In: ZEUS, pp. 1–8 (2018)
62. Wolff, E.: Microservices: Flexible Software Architecture. Addison-Wesley Professional, Boston (2016)
63. El-Sheikh, E., Zimmermann, A., Jain, L.C.: Emerging Trends in the Evolution of Service-Oriented and Enterprise Architectures. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40564-3
64. Sampaio, A.R., et al.: Supporting microservice evolution. In: IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 539–543. IEEE (2017)