SOCA (2014) 8:129–158
DOI 10.1007/s11761-013-0153-3

ORIGINAL RESEARCH PAPER

A survey of methods and approaches for reliable dynamic service compositions

Anne Immonen · Daniel Pakkala

Received: 13 June 2013 / Revised: 7 November 2013 / Accepted: 30 December 2013 / Published online: 22 January 2014

Abstract An increasing amount of today's software systems is developed by dynamically composing available atomic services to form a single service that responds to consumers' demand. These composite services are distributed across the network, adapted dynamically during run-time, and still required to work correctly and be available on demand. The development of these kinds of modern services requires new modeling and analysis methods and techniques to enable service reliability during run-time. In this paper, we define the required phases of the composite service design and execution to achieve reliable composite service. These phases are described in the form of a framework. We perform a literature survey of existing methods and approaches for reliable composite services to find out how they match with the criteria of our framework. The contribution of the work is to reveal the current status in the research field of reliable composite service engineering.

Keywords Composite service · Service architecture · Reliability evaluation · Reliability-awareness

1 Introduction

Today, the need for advanced services is rapidly increasing, as consumers' needs become more demanding and complex, most notably as a result of the growth in demand for mobile devices and ubiquitous computing environments. Composite services [1] aim to respond to today's challenges, as they combine several atomic services to form a unique service composition that provides a solution to these complex demands. Service-oriented architecture (SOA) [2] represents an appropriate architectural model for composite services, enabling, even dynamically, combinations of a variety of independently developed services to form distributed, software intensive systems.

Composite service developers form a new kind of service consumer group, as they act both as a service consumer and a service provider, composing services into one ensemble and then providing it to consumers as a single service. The composite service must fulfill both the task goals of the service from the composite service consumer's viewpoint and the business goals of the composite service provider. When well planned, a composite service is once designed and then dynamically managed and evolved during run-time. These dynamic capabilities require that the design must take into account the run-time requirements to dynamically discover, select, and bind available services and the capability to adapt due to changes.

Service reliability and its verification become a prerequisite as these new kinds of modern services are present in our everyday life affecting several everyday functions and situations. Service reliability can be defined as the probability that the system successfully completes its task when it is invoked ("reliability on demand") [3]. For composite service consumer, this means that the service works correctly and without interruptions and is available when required. For a composite service provider, reliability has a broader meaning; the provider is able to 'trust' that the service fulfills the requirements and works as expected. New modeling and analysis methods and techniques are required to enable to compose and manage reliable service compositions at run-time.

In this study, we take the viewpoints of composite service developer and composite service provider on achiev-

A. Immonen (✉) · D. Pakkala
VTT Technical Research Centre of Finland,
P. O. Box 1100, 90571 Oulu, Finland
e-mail: anne.immonen@vtt.fi

D. Pakkala
e-mail: Daniel.Pakkala@vtt.fi

ing reliable composite service. The role of the composite service developer consists of different stakeholders, such as requirement engineers, architects, evaluators, coordinators, managers, and financial controllers, which influence the service at design time. The service provider influences the service at run-time, having responsibilities of making the service available for consumers and managing the service availability. Since all stakeholders have different roles in service life cycle, they also have interests in reliability and responsibilities in achieving it. In addition to traditional service reliability engineering, several new challenges arise in this new setup:

- How to select reliable services for a composite service? Reliability of composite service is dependent on reliabilities of its atomic services. The composite service provider should be able to analyze atomic service reliabilities, compare different selections, and be ensured about the service actual reliability before service binding. Reliability analysis assists in building confidence to the technical capabilities of the service. However, reliability can have “softer”, non-technical forms that manifest themself between things, people, or organizations. Reputation relies on the aggregation of experiences with the service or service provider. Further, trust is gained from a person’s or organizations’ own experiences with a service or service provider. These other forms of reliability also have a great influence on service selection.
- How to ensure reliability during composite service run-time? In addition to selecting reliable services, composite service providers must be able to verify the reliability of the selected services and the composite service during run-time. This requires that the service execution is monitored. The monitoring unit collects and stores statistics of the quality characteristics of services at regular intervals. Based on this collected statistics, run-time reliability analysis is required to analyze both the atomic service and the composite service reliability by calculating the quality metrics.
- How to manage reliability? Reliability analysis as such is not enough for composite service providers to ensure service reliability. If detecting undesirable changes in reliability of services or the reliability requirements are not being met, the service system must have a plan how to act based on the analysis results. The service system must be able to adapt itself due to changes that occur in important characteristics of a service (e.g., reliability, reputation, and provider’s trustworthiness) to maintain the required reliability. To enable this, the composite service provider needs a comprehensive service composition and monitoring architecture with decision making logic. Therefore, the dynamic nature and the run-time reliability requirements of composite service need to be taken into account already in requirements engineering and design phase.

Implementation of a composite service may be deployed inside one device or into several servers across a network. Furthermore, we believe that the future’s composite services will apply increasingly the cloud computing environments [4], which provides software as a service (SaaS) through a program interface, hiding the complexity of implementation from the consumer, allowing the composite service to scale to the amount of consumers. In any deployment and in any computing environment, the reliability of composite service must still be ensured and managed. To engineer reliable services, the composite service developer requires means, methods, and techniques for specifying reliability requirements, bringing reliability requirements into design, selecting services based on reliability requirements, and analyzing and verifying reliability (both at the architectural level and on run-time).

There already exist several studies in the literature considering reliability analysis, service selection, and quality adaptation. In this study, we discuss the essential issues that the composite service developer should consider when engineering reliable composite services. We then define a domain and implementation independent evaluation framework with evaluation criteria derived from the issues in reliable composite service design and execution. We perform a literature survey of existing approaches and methods for reliable composite services, which are not restricted to any specific domain, and examine how the approaches in the literature respond to these criteria. Thus, the purpose of the framework is to assist in revealing how the criteria are taken into account in the current approaches in the literature, and what is the status of the research literature considering each of the criteria. We select nine approaches for further examination and evaluate how these approaches consider and implement the discussed criteria. Our literature survey also reveals the current status in the research field, exposing the potentials, shortcomings, and development targets.

In the next section, the essential issues in composite service design and execution are discussed. Section 3 introduces our evaluation framework. Section 4 describes the results of our literature survey, which include the evaluation of the selected approaches and the discussion about the other related literature for each of the framework criteria. We discuss the emerging issues of our research in Sect. 5, and finally, in Sect. 6, we conclude our work.

2 Overview of reliable composite service design and execution

Reliable composite service design and run-time consist of several phases. Figure 1 describes these phases from the composite service developer (design time) and provider (run-time) viewpoints. Requirements engineering, service

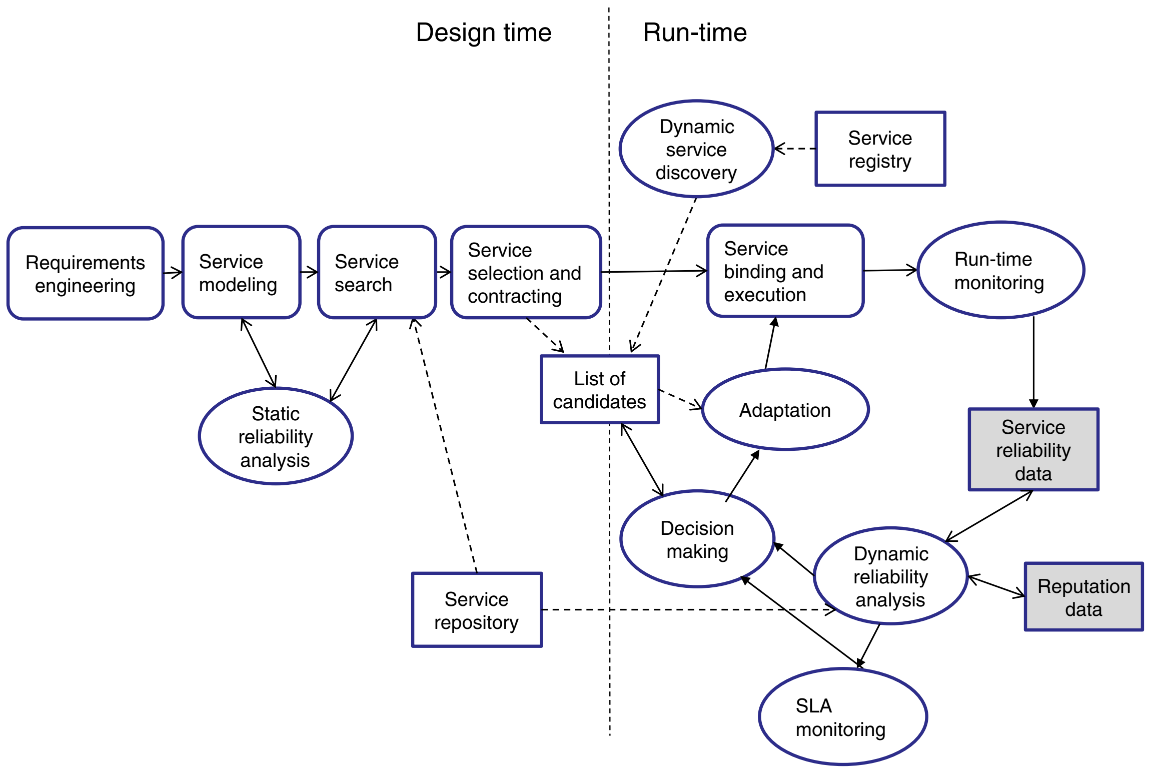

Fig. 1 Reliable composite service design and run-time modeling

modeling, service search, service selection and contracting, and service binding, and execution are the traditional phases in the composite service design and execution. The required data sources for the traditional phases (service repository and service registry) are described in Fig. 1 using rectangle shapes. The new required phases to achieve and manage reliability are described using the ellipse shape to separate them from the traditional phases. New required data sources are described using gray rectangle shapes.

We assume that the composition service developer is an organization that develops a composite service that fulfills some complex consumer's tasks. The developer specifies the functionality and quality of the service according to anticipated, foreseen, or prespecified demand, focusing on achieving economic goals by providing the service for multiple consumers. Reliability requirements are achieved using a formal requirement engineering (RE) method and one or more requirement elicitation techniques. Reliability requirements contain both measurable, quantitative requirements and qualitative requirements that end up to some design decisions. Measurable reliability requirements for a service must be expressed using metrics, with which the target values can be defined. In reliability analysis, these values are used to verify the fulfillment of requirements. The means to achieve and manage reliability during run-time must be taken into account in design decisions and must therefore be noted already in requirements engineering phase. Therefore, reliability requirements must cover the requirements for context-awareness, reliability-awareness, and self-adaptiveness. From service-centric viewpoint, context-awareness is here restricted, meaning only that the dynamic service discovery detects if new services (possible with better reliability) become available, whereas reliability-awareness means that the service is aware of its current reliability. Self-adaptiveness means that software is expected to fulfill its requirements at run-time by monitoring itself and its context, detecting changes, deciding how to react, and acting to execute decisions on how to adapt to these changes. This requires both context-awareness and reliability-awareness.

Service architecture is a set of platform and implementation technology independent definitions of structure, functionalities, data, management, and control of services within a service-oriented system. Service architecture should be modeled in a way that assists in reliability analysis and to compare different service selections. Ideally, service architecture describes the conceptual service in a form of a service template or as an abstract, conceptual level architecture into which the suitable services are searched. The reliability requirements must be brought into the architectural design in a formal way. Usually, they emerge in a form of design decisions and required qualities for services to be searched. After finishing service architecture, the composite service developer must perform the feasibility study, inspecting carefully which part of the system it is rational to implement using ready-made services offered by several service providers. Since there exist several services that embody the same functionality, the key in service selection is to find services that fulfill the non-functional requirements best. For reliable composite service, the services for the service architecture are searched mainly based on their reliability information. Services are commonly searched from a service repository, which is a development time registry of human understandable (may also be machine interpretable) service descriptions that can be utilized in software development. The service repository contains usually static information about the reliability of services that is subjective (i.e., provider's advertised service information). If possible, the actual, dynamic reliability information about the services should be obtained from the dynamic environment, possible from the dynamic data storages. The trustworthiness of the information must be ensured.

Reliability analysis emerges already in the design time. The static reliability analysis can be pure architectural level analysis, which is done in service modeling phase. In that case, the analysis is done using the estimated probability of failure values of services and behavioral description. This kind of predictive analysis provides, among others, feedback to service design and helps to detect critical services. The static analysis can be performed also in service searching phase using the obtained reliability values (static or dynamic, if available). Services with low reliability decrease the reliability of the whole composite service, and that is why reliability evaluation is extremely important within composite services. The analysis helps to compare the services and the different service selections before service binding.

Based on reliability analysis, the services are selected and possible contracts [including service level agreements (SLAs)] are made with service providers. Service reliability is a strong justification for selection of reliable composite service, but it is not always enough. The weak reputation of the service provider is a good reason not to choose a certain service. The service reputation itself should be a justification for selection, since it reveals how it has been experienced. Trustworthiness, if reachable, tells how the promised quality meets the actual quality, revealing also the trustworthiness of the service provider. Within devices, the status of the device has a great influence on its selection. Finally, selection is strongly affected by the reliability of the service at the time of selection. A highly reliable service can be sometimes unreliable or even unavailable. That is why the service dynamic reliability information should contain both the most recent and average reliability. The SLAs with the service providers are a good way to save the composite service provider side. In SLA, numerous service reliability metrics are defined, and their values are negotiated. In design time, for each service, several candidate services are searched. The list of candidate services should be established from where the suitable services can be selected when services in used need to be replaced. The SLA contracts with the candidate service providers should also be negotiated beforehand so that there will not be any breaks in composite service execution when replacing services.

After service selection, the services are bind and the composite service execution begins with the help of service registry. Service registry is a run-time registry of machine interpretable service descriptions, which includes all the services that are available at that moment. The service discovery should still continue dynamically during run-time (i.e., context-awareness). In the run-time, the dynamic service discovery updates the lists of candidates. The SLAs still need human inference and should therefore be prenegotiated regularly. The executed services are monitored (i.e., reliability-awareness). The monitored data are stored in dynamic data storage for reliability analysis purposes. Within devices, status should also be monitored during service execution. Consumers may be provided some kind of system to vote or evaluate the service he/she just has been consumed. These data can be used for reputation calculation for service or service provider, and also has to be collected and stored in dynamic data storage for analysis purposes. Dynamic data storages store two kinds of information. The raw, monitored, or collected data are used for analysis purposes only. The calculated service reliability or reputation values are also stored in data storage. The stored information should include both average and the most recent values. Ideally, this information could be available for anyone searching for dynamic reliability information about the services.

Dynamic reliability analysis calculates actual reliability values for services. After a certain time unit, it obtains the data from the dynamic data storages to calculate dynamic service reliability and service reputation. For trustworthiness calculation, the analysis requires also the service provider's advertised service reliability information. SLA contract is being monitored based on the reliability analysis. Decision making unit is responsible for making decisions to ensure and manage reliability by performing dynamic quality adaptation (i.e., self-adaptiveness). Usually, this means that the unreliable services are being replaced if needed. The unit has to clarify whether the purpose is to obtain the optimal reliability, or is it enough that the reliability values are inside a certain value range. If optimal reliability is selected, the decision making unit checks regularly the list of candidate services to inspect whether services with better reliability are detected. If true, the services in use are replaced with services with better reliability values. In other case, the list is used if the reliability analysis shows that some services in use are unreliable or the reliability of the composite service drops below the required level. The decision making unit maintains and updates the list of candidates according

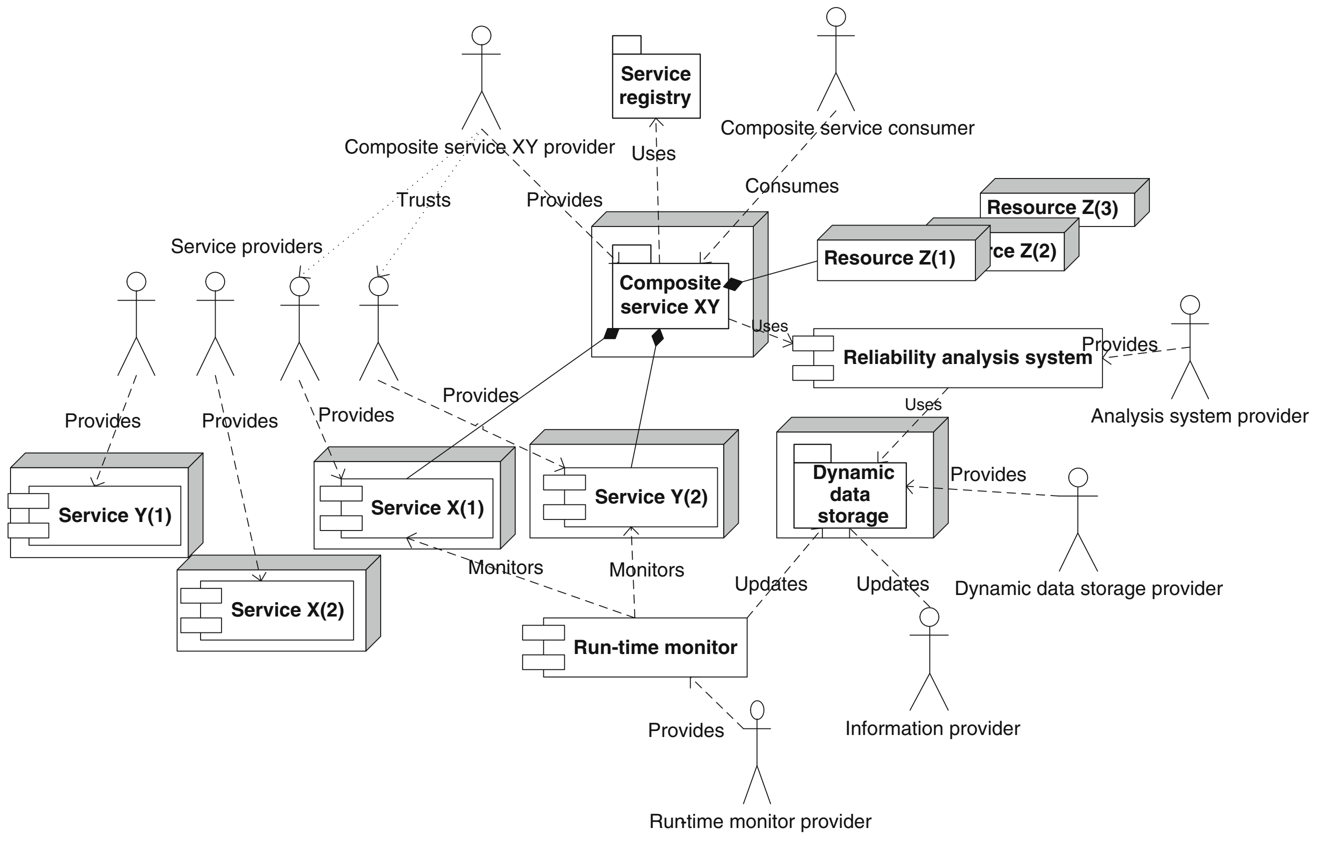

Fig. 2 Composite service in its deployment environment

to its own criteria for service selection, keeping services in list in order of superiority. In addition to reliability analysis, the SLA checking may also cause the decision to adapt.

Figure 2 describes the composite service in its deployment environment. We do not want to limit the service definition of our work to include only software services; thus, a service can be defined as including both software services and device services. Software service is “a function that is well defined, self-contained, and does not depend on the context or state of other services” [5], whereas a device can be defined as easily accessible computing devices that assume many different forms and sizes, able to communicate with each other and act “intelligently” [6]. In SOA, the services and devices are considered as independent resources provided independently by multiple service providers. A device provides device services to software services, whereas a software service may require a device to implement its functionality. A composite service is here understood according to the definition in OASIS reference architecture for SOA [1], thus being visible to consumer as one service and being a composition of several atomic services. A composite service is formed by composing dynamically the available resources.

The functionality of composite service XY in Fig. 2 consists of two software services: Service X and Service Y, and one device service: Resource Z. To fully operate, composite service XY requires a comprehensive composition and monitoring architecture that includes the required logic for reliability monitoring, context monitoring (i.e., new services appearing), change detection, and decision making and executing. It uses service registry to discover services. Atomic services are located across the network on different network nodes and provided by different service providers. For each selected atomic services (Service X, Service Y and Resource Z), there exists alternative services in the network that implement the same functionality. These services can be selected instead if the reliability of the selected services decreases. Composite service provider is the actor that provides the composite service to service consumers. Composite service provider must maintain a network of reliable service providers, since the trust and reputation of the service provider may be one of the service selection criteria. Service consumers can be human user via user interface or another service.

To guarantee reliability, the composite service must include “means” to achieve and manage reliability. This includes the means for context-awareness, reliability-awareness and self-adaptability. Run-time monitoring system is required to monitor the service execution. Reliability analysis system is required to analyze the reliability of the composite service and its atomic services. Both the monitoring system and the analysis system may be provided by the composite service provider himself or third-party system providers as in Fig. 2. Dynamic data storage is required to store the dynamic reliability information. These data are updated by the monitoring system and by other information providers, such as other monitoring systems, service agents and user evaluations that offer valuable information about the actual use of the services. The data storage provider can be composite service provider or a third-party data storage provider.

3 Evaluation framework

This section introduces our evaluation framework. The criteria of our framework are derived from the issues discussed in Sect. 2. To be applicable to different context, the criteria are purposely described in quite high level. The approaches are inspected and discussed from the following viewpoints:

- Context of the approach: the approaches are introduced and their main content regarding reliability is examined
- Reliability requirements and service architecture design: the early composite service development phases are examined
- Reliability analysis: the reliability analysis methods are inspected
- Decision making logic: the decision making involving service selection and quality adaptation is inspected
- Composition and monitoring architecture: the means to achieve and manage reliability are examined
- Applicability of the approach: inspection of the approach as "a method to be applied"

Each of the viewpoints is discussed in more detailed in the next sections.

3.1 Context of the approaches

The purpose of this section is to examine the background of the methods/approaches. In addition to methods' main characteristics, the compared issues include:

- composite service definition: how does the approach define the term composite service
- application area: where can be approach be applied
- main content: what is the main focus of the goal of the approach
- expected benefits: what can be achieved by using the method/approach

Currently, in the context of Web services, a service is often defined to include only software services, whereas in the context of pervasive/ubiquitous systems, the service stands for both software services and device services. The dominant standards to publish and discover services, such as Web Ontology Language for Services (OWL-S) and Web Services Description Language (WSDL), do not enable the description of quality of service-related information. Due these restrictions, many extensions to these standards have been developed [7–10] that extend OWL-S.

Service-oriented architecture and composite services have been commonly applied within Web services [11–13]. Composite Web service includes multiple Web services as part of a composite service, providing its service to consumers through single interface. Web services seem to be more mature technology, since it has well-defined technologies and standards. Also, the number of efforts at standardization of semantic descriptions and quality description in connection with Web services reveals its popularity and potential. The common interest in Web services is also revealed by the number of literature surveys of the approaches considering Web service selection and composition [12,14–17]. Recently, the idea of composite services has been increasingly applied also in the context of ambient intelligence (AmI) environments [18–20], which obeys the guidelines of pervasive and ubiquitous computing. Pervasive computing [21] refers to easily accessible computing devices connected to a ubiquitous network infrastructure. Ubiquitous computing [22] allows consumers to complete simple or complex tasks using the available resources; the environment enables that the connectivity of devices is embedded in a way that it is always available and invisible to consumers. Some of the AmI systems still employ Web service technologies and standards, and some of them relate to services of different technologies, such as UPnP [23] or OSGi [24].

The main focus of the methods/approaches inspected in this survey is to enable reliable composite service. The approaches may still have different ways and methods to achieve the main focus. Therefore, the main emphasis of the approaches can vary between reliability analysis methods, service selection methods, monitoring architecture, etc. Also, the expected main benefits of the methods may vary due to different main contents.

3.2 Reliability requirements and service architecture design

The purpose of this section is to examine how the requirements affecting reliability are engineered, modeled and brought to the architecture design. The inspected issues include:

- reliability ontology: the structure and description format of reliability-related information
- reliability requirements: the content, engineering process and description format of requirements affecting compos-

ite service reliability and the means to achieve and manage reliability
- transforming reliability requirements into architecture design: description of how the requirements affect the design
- composite service architecture description: the description of service architecture based on the requirements engineering.

3.2.1 Reliability ontology

The use of ontologies is one of the key issues in the requirement engineering and architectural design enabling complete, consistent, and traceable requirements and design decisions. Furthermore, ontologies are a formal way to express and share knowledge about the service in question. Requirements ontology describes the structure of what and how requirements should be described, and guides the elicitation process with stakeholders. Quality-of-service (QoS) ontology consists of concepts and their relationships related to quality engineering, containing quality metrics, i.e., unit of measurements and quantitative ranges. QoS ontology facilitates the matching of required QoS properties with provided QoS properties. Reliability ontology describes the structure and description format of reliability-related information, consisting of concepts and their relationships with reliability engineering. The capabilities of devices, such as device quality and other attributes, should also be described in the form of ontology. Despite ontologies, quality policies [25] can be used to generate quality objectives, and they also serve as a general framework for action. Policy-based computing means that the computation constraints are specified and analyzed separately, and enforced and evaluated at run-time. To simplify, a policy is a collection of alternatives, which consists of assertions, each of them representing one requirement, capability or behavior of a service. On the composite service provider's side, policies can be used to describe the required quality properties and the actions in service selection. At the atomic service provider's side, policies are used to describe and represent the provided quality properties of services in a standardized way.

3.2.2 Reliability requirements and means for achieve and manage reliability

Requirements describe the features or capabilities that a service must have. Requirements must be documented, measurable, testable, traceable and defined in such a detail level that is required for service design or for searching suitable services. Requirement can be commonly classified into functional, non-functional and constraints. Functional requirements describe the behavior of a service that support the tasks or activities of the user. Non-functional requirements describe the qualities of the system, such as reliability, performance and security. Constraints are those characteristics that limit the development and use of the service (design, implementation, interface or physical requirements). Reliability requirements must consider both quantitative and qualitative requirements. The quantitative requirements are described using metrics and are verified using reliability analysis. The qualitative requirements are non-measurable, ending up to some design decisions. The qualitative requirements thus contain the "means" to achieve and manage reliability (i.e., the means for context-awareness, reliability-awareness and self-adaptability).

Commonly, a lot of attention has been paid to technical solutions for making service-oriented systems, such as service discovery, selection, invocation, composition and interoperability [13,26,27]. Less attention is being invested in requirement engineering for services, although the poor and inadequate requirement engineering has been recognized as one of the main reasons why software development projects fail. Most of the traditional RE methods in the literature concentrate on functional requirements. Although some approaches on non-functional requirements (NFR) engineering exist, there is still no agreed or clear pathway on how to derive NFR from business goals, how to transform them to design and how to verify whether they are met. Also, the transfer from the RE tools to architecture design tools is weak when considering requirements traceability. Non-functional attributes exist on the composition level as well as on the atomic service level, which complicates their definition, representation and analysis. Very often it is also difficult to understand satisfaction criteria of non-functional requirements.

The existing requirement engineering methods can be applied also in the case of services, but they are inadequate as such. Service orientation brings several new challenges into requirement engineering. These challenges are derived the special features of service-oriented computing, but also the goals and business objectives of the service providers. Although service should be developed to satisfy the needs of the service consumers, the service may have a significant impact on providers as well. The conflicting service needs of different stakeholder groups must be communicated and negotiated. The real challenge is that the actual service consumer is not usually known in RE phase or cannot be reached for requirement elicitation. Service development needs to deal more with uncertainties, which must be recognized and anticipated by the requirement engineering phase. For example, decisions affecting the service capacity, i.e., number of requests the service can handle without affecting its reliability and availability, must be done before knowing the amount of future consumers. Furthermore, the final deployment environment may not be known in RE phase, but the boundaries are set during the RE phase. The roles and responsibilities of services are also difficult to define, since the final composition of services is not always known in RE phase, and service roles can vary as the composition of services changes. The requirements for reliability-awareness, context-awareness and self-adaptability also bring new challenges. According to design for adaptability guidelines represented in [28], entirely new actor, adaptation designer, needs to be introduced to define adaptability concepts. In addition, to be able to monitor itself and the context, detect changes, decide how to react and act to execute decisions, the self-adaptive software requires a management system that implements the decisions in a controlled way.

3.2.3 Transforming reliability requirements into service architecture design

After requirements engineering, the composite service architecture must be specified by dividing the specified system tasks into abstract, conceptual level services. The requirements are brought to the architecture by mapping them onto the architectural elements, when the fulfillment of requirements becomes the responsibility of the services. The quality requirements are transformed into design decisions, i.e., the properties that the composite service has to have. The conceptual services are initially defined in a service template form or as a conceptual architecture; the concrete services for each conceptual service are determined either before run-time or dynamically at run-time. The service architecture should be described in a formal way, e.g., using Unified Modeling Language (UML) [29], which is a standard and widely accepted modeling language. As such, the UML is inadequate to express quality in design, but it can be extended by specific profiles to support certain quality aspects. Profiles for Schedulability, Performance and Time [30], and for modeling the QoS and Fault Tolerance [31] have already been suggested.

3.2.4 Composite service architecture description

A service-oriented architecture is a collection of services that communicate with each other by exchanging data or by coordinating some activity with two or more services. Service architecture is the first asset that describes the composite service as a whole. For each composite service, the service architecture is a unique structure of its building blocks, i.e., services, and a description of how data, management and control flow between these services. Architecture can be described with different details depending on the purpose, e.g., for the reliability analysis, the architecture should be described in a way that enables the analysis performed in the analysis phase. Architectural description can also have different abstraction levels. The conceptual level means delayed design decisions concerning, for example, functionality. Concrete level refines the conceptual designs in more detailed descriptions.

3.3 Reliability analysis

The purpose of this section is to examine how the reliability of the composite service is analyzed, both at the design time and run-time. The examined issues include:

- analysis level: applicability of the analysis to different phases of composite service development and execution
- data source for the analysis: the data source for the analysis and the trustworthiness of the source
- analysis technique/method: the complexity and applicability of the reliability analysis method
- analysis output: what is achieved with the analysis.

3.3.1 Analysis level

Composite service reliability analysis can be divided into static and dynamic analysis. Static analysis is architectural level analysis, performed before service binding and execution. In the ideal situation, architectural level reliability analysis can be performed purely on the conceptual abstraction level prior to making any implementation-related decisions, such as selection of services. This kind of analysis has several benefits, such as the reliability problems can be solved more easily, and modifications are easier and cheaper to implement. The analysis also helps to detect the most used and therefore critical services of the system; special attention should be paid to how to ensure the reliability of those services. The effects of the design decisions of system reliability can be detected beforehand, and the required design decisions to ensure reliability can be made based on the analysis. This kind of predictive reliability analysis can be performed using the estimated probability of failure values of services and behavioral descriptions. The most used way to perform static reliability analysis is in service selection phase by analyzing service reliability with selected services before system deployment and execution. This kind of analysis helps to compare different service candidates before final service selection and binding. The analysis can be performed by using reliability information available about the services (static or dynamic).

Dynamic analysis is run-time analysis that occurs during composite service execution. For dynamic reliability analysis, composite service provider needs to achieve dynamic, actual information about the quality of the services through active monitoring of service execution and possible feedback from the previous service consumers. As a result, the analysis projects the probability of failure of individual services that form the composite service, and also the probability of failure of the composite service with these selected services, enabling also detection of changes in service quality. The dynamic analysis also provides information about the actual usage profile of the system.

3.3.2 Data source for the analysis

Acquiring data from the candidate services for reliability analysis is also a great challenge. Usually, service providers make this information available whether by advertising it or providing an interface to access the data. Either way, the quality information is subjective and static. Different approach is to collect quality information by utilizing the earlier consumers of a service. Earlier consumers provide objective insight into the service and can therefore provide valuable information for service trustworthiness and reputation evaluation. Another approach is to trust in a third party to collect information or to rate certain service providers. This approach is expensive and inflexible in a dynamic environment where changes occur constantly. The safest way to collect objective data is the use of monitoring systems that actively observe the execution of services, collect statistics and calculate the actual, realized values of the quality metrics of the service. Since reliability is generally an execution quality of software, monitoring helps to achieve the actual, dynamic quality of the services.

3.3.3 Analysis method

The first service-oriented reliability analysis methods appeared in 2003–2005 [32–34] and several promising approaches have been suggested since. Traditional reliability analysis methods can be roughly classified into quantitative and qualitative methods. The quantitative reliability analysis computes the failure behavior of a system based on its structure in terms of composition (i.e., services and their interactions), and the failure behavior of the services and their interaction. Methods employing quantitative techniques are further classified into state-based, path-based and additive models [35]. State-based models use the probabilities of the transfer of control between components to estimate system reliability, whereas path-based models compute the reliability of composite software based on possible execution paths of the system. Additive models address the failure intensity of composite software, assuming that the system failure intensity can be calculated from component failure intensities. The qualitative reliability analysis consists of reasoning the design decisions (e.g., fault tolerance and recovery mechanisms) and their support for the requirements. By analyzing and reasoning about one architectural solution, qualitative analysis provides assurance that the requirements have been addressed in the design.

Traditional reliability analysis approaches can be applied in the context of composite services, but only at the architectural level, where the analysis is based on static information. The suitability of architecture-based quantitative reliability analysis methods for component-based software is analyzed in [36], according to which, although there are several methods available for reliability analysis, they still have serious shortcomings that restrict their application in the industry. These shortcomings include lack of tool support, weak reliability analysis of atomic software components, and weak validation of the methods and their results. The architecture-based reliability models cannot be used during service execution, since they assume that the reliability of components is known or can be estimated, the architecture is stable, and the connections among the components are reliable.

3.3.4 Analysis output

For composite services, there are two levels of reliability: atomic service reliability and composite service reliability. Atomic service reliability is the reliability of the service as an independent unit. Drawing the line and isolating the service from its surroundings for reliability analysis are still hard, since service reliability is affected by the reliabilities of the services it requires to operate and also by the reliability of the context, i.e., the operating environment. The reliability of the atomic service is often advertised by the service provider, and the composite service provider is forced to trust its validity. Since reliability is an execution quality, i.e., observable during run-time, the actual atomic service reliability can be determined only during system execution. The composite service reliability is affected by the reliability of each service in the composition, the reliability of the operating environment and the service interactions in the form of usage profile. Usage profiles are of great concern in the frequency of executing each service and each interaction between services, and therefore, they form a complex challenge when analyzing composite service reliability. When a composite service is composed dynamically, usage profiles will be unknown beforehand and can be observed only during execution. As an execution quality, the impact of faults and low reliability of atomic services on composite service reliability differ depending on how the system is used.

3.4 Decision making logic

In this section, the purpose is to examine how and based on what information the decisions in service selection and adaptation are being made. The inspected issues include:

- type of data for decision making: description of the type of data, which is used in decision making
- service selection criteria: description of the criteria for decision making

- service selection process: description of how the service is selected
- quality adaptation: description of how the quality adaptation occurs.

3.4.1 Type of data for decision making

Service selection policies can be divided into static and dynamic policies. Static policies are based on the static, stable, unchangeable information about the services. This includes, for example, license issues, technical and functional descriptions, and any information that has been made available by the service provider, including quality information. Dynamic policies, on the other hand, take into account the dynamic information about the service, e.g., the current status and the actual, dynamic quality information. Only the dynamic, objectively observed information can be used in reliable service selection.

Dynamic information may also have two separate aspects: average and the most recent. Average quality information is the long-term, measured quality of a service past execution up to the present, whereas recent quality information concerns the current quality of the service, measured in the most recent time period. Highly reliable service can sometimes be less reliable through a certain cause, such as malicious attacks, as a sudden change in service execution can change the values for the metrics temporarily. In some cases, the most recent values matter more when the request for a service arrives. In those cases, quality information should be updated in real time.

3.4.2 Service selection criteria

There exist several studies in the literature about what information about a service is required to enable selection of reliable services. At the moment, there exists no agreement about quality attributes. The work in [37] represents the W3C Working Group's consensus agreement as to the current set of requirements for the Web services architecture. In addition to [37], in several other cases such as [8,38–43], reliability issues are considered to be relevant, after which often follow availability, security and performance. Very often also cost, i.e., the price for using the service, has been included as a part of selection criteria [39,41,42,44], as well as the service capacity [37,41,43].

Besides technical quality, selection may also be based on softer, non-technical quality properties. Non-technical reliability properties are indirectly related to service, helping to build "trust" to the service. Trustworthiness is defined as the degree of confidence that the software meets a set of requirements [45]. These requirements can involve any critical requirements relating to service functionality and quality, but also to non-technical properties, such as licensing and characteristics of the service and service provider. The concept of reputation is closely linked to trustworthiness, meaning "what is generally said or believed about a person's or thing's character or standing" [46]. User experiences, feedback about the actual use of the service, and general opinion build the reputation of the service, and often also the reputation of the service provider. Trustworthiness can be achieved, for example, by comparing the advertised quality provided by the service provider with the actual quality. Reputation-based approaches are usually based on user feedbacks and/or system/user monitoring. For device services, information about dynamic quality is required as well, but also information about its technical details. Furthermore, within devices, the status of the device has a great influence on its selection. Status information can vary depending on the service and the service domain. For example, the location of the service compared with the requestor must be known when the selection is based on geographical distance. The amount of current CPU or memory usage and free disk/memory space can effect selection and so can data transfer speed and reliability of transfer. Therefore, the status of services at the time of selection should be checked and also monitored during service execution.

3.4.3 Service selection process

Reliable service selection can be roughly divided into three phases: service search, dynamic information gathering and selection. At first phase, available services are searched traditionally, based on static information, using, for example, the standard publishing and finding techniques. The required policies are compared with the provided policies, and the candidate services are selected for further comparison. In the second phase, the actual, dynamic information about the identified candidate services are gathered. The reliability information is collected, for example, to dynamic data storage, through an active monitoring of the service execution or through user feedbacks. The information is used to calculate values for service reliability metrics. These values denote the actual reliability of the service. For device services, the status of the candidate and currently available services is being checked. In the third phase, the best suitable service is selected. The requirements are compared with the achieved dynamic reliability data, the status information, preferences and priorities. Matching engine/algorithm is required to compare the required quality characteristics with service dynamic quality policies. The decision making unit must have rules how the selection is made using these criteria, and what are the priority and preferences of the selection criteria. Trade-offs has to be made, for example, is the better reputation better than distant location of the service. The reliability analysis of the composite service can be done with real reliability values. The analysis gives estimations about the reliability of the composite service with these selected services. Services can still be changed, and the reliability of the different alternative compositions can be checked before the final service binding.

3.4.4 Quality adaptation

The classification of adaptability properties by Horn [47] serves as the de facto standard in the domain, including self-configuring, self-healing, self-diagnosis, self-optimizing and self-protecting. Self-configuring, self-healing and self-protecting assist also in achieving reliability. These properties require self-awareness and context-awareness to operate. Self-configurable composite service can configure itself automatically in response to changes. With the help of self-configuration, the service composition can be configured when services with better reliability are detected or the reliability of the services in use declines. The purpose of self-configuration is whether to achieve optimal reliability or the reliability that satisfies the requirements. Self-healing maximizes reliability of the system by discovering, diagnosing and recovering from failed services. The means to attain dependability [48] are applicable in anticipating potential problems and acting to prevent failures. The purpose of fault prevention is to prevent the occurrence or introduction of faults. Fault tolerance tends to prevent systems from failing in the presence of faults, consisting of error detection, and system recovery through error and fault handling. Fault removal indicates a reduction in the number and severity of faults during the use of a system through corrective or preventive maintenance. Fault forecasting estimates the probability of faults and failures according to the current and future behavior of the system, as well as the consequences of projected faults. With self-protection, a system prepares itself to defend from malicious attacks by detecting security breaches and recovering from their effects.

Six questions for eliciting requirements for a self-adaptive system can be identified [49]: where is the need for a change; when a change needs to be applied, what needs to be changed; why is a change needed; who should implement the change; and how the adaptable artifact is changed. From the architectural point of view, dynamic adaptation requires special architectural elements that must be taken into account in the composite service RE phase. Run-time monitors are the "standard" solution to assess the quality of running applications. A monitoring system must be available to monitor the service execution, and a collection mechanism is required to gather user feedback, ratings, etc. The monitor and collection mechanisms store statistics of the quality characteristics at regular intervals, which are then used in reliability analysis by a calculation unit that calculates the quality metrics at regular intervals to maintain dynamic quality information on the services. The decision making unit is responsible for detecting when and where the change is required and what needs to be changed and how. The decision making unit compares the results of the reliability analysis against the composite service requirements, and also the history data, to detect when a response to a change is required. After detecting where the change is required, the decision making unit decides what needs to be changed and how the change should be implemented. The decision making process should occur during run-time and should not require human inference. In addition to adaptation, the decision making unit must have rules determining how the selection of replacement services is to be made. The decision making unit must have strictly defined selection criteria, and the priority and preferences of the selection criteria. Trade-offs must be made, for example, with respect to the higher reputation over distant location of a service.

3.5 Composition and monitoring architecture

In this section, the means of the approaches to achieve and manage reliability are examined. The required elements of the composition and monitoring architecture are inspected in more detail. These issues include:

- dynamic list of candidate services: a back-up plan in the case of failure or decrease in quality
- run-time service monitoring: the monitoring activity of the service execution
- SLA monitoring: monitoring of the contract between atomic service providers and composite service provider (here in a role of a service consumer)
- feedback collection system: how the data are achieved for trustworthiness and reputation evaluation.

3.5.1 Dynamic list of candidate services

In the case of failure or unfavorable change, a composite service must have a ready-made plan how to act to guarantee the continuous service execution. For reliable composite service, this back-up plan is usually a list or a pool of candidate services, i.e., for each abstract service, there is a possibility to bind it to a set of functionally similar concrete service. Ideally, the reliability of these candidate services has been verified beforehand. By keeping services in list in order of superiority, the next service is already preselected when a service in use needs to be replaced. This list should be updated dynamically, i.e., it should always contain the best available candidates.

3.5.2 Run-time service monitoring

Recently, quality awareness has been seen as a persistent challenge in service computing [13,26,50]. Some attempts have been made to standardize service monitoring; however, none of them have been commonly adopted. Web Services Distributed Management (WSDM) [51] is a promising OASIS standard for controlling and monitoring the status of other WSDM-compliant services. The specification defines how to represent and access the manageability interfaces of resources as Web services, how to manage Web services as resources, and how to describe and access that manageability. The earlier studies of fault monitoring approaches and tools, such as [52], reveal that although there are numerous approaches for implementing monitors, most of the tools are for research use only. Thus, although there is an enormous demand for monitoring approaches, the concept of how to implement it is not mature. The huge amount of papers concentrating on service systems monitoring reveals that there is a great demand for execution-level quality verification.

Commonly, the monitoring-based approaches monitor only the selected services. Monitoring all available candidate services, such as in [40], seems unsuitable since a large amount of services may exist. When monitoring only the selected services, the dynamic quality of candidate services cannot be verified when the selected services need to be replaced. A good solution could be a dynamic list of suitable candidate services, such as in [53], which quality is monitored and updated regularly.

3.5.3 SLA monitoring

A SLA is a legal negotiated agreement between a service provider and a consumer, defining the "level of service" for each area of the service scope. It describes the agreed-upon terms with respect to quality of service and other related concerns, such as price, being also a guarantee of the promised quality. Further actions and possible sanctions have been negotiated in SLA as, for example, if the requirements are not being met. Web Service Level Agreement (WSLA) [54] focuses on specifying and monitoring SLAs for Web services, but it does not address the modeling and management of the QoS of composite services.

In the case of SLAs, the composite service provider acts as a service consumer, requesting and using atomic services as a part of his/her composite service. To be useful for composite service provider, the use of SLAs requires the SLA monitoring. If the SLA agreement is not met, the composite service developer must have the means for selecting alternative services to guarantee the availability and reliability of his/her composite service. For these alternative services, the composite service provider needs a list of trustworthy service providers and prenegotiated SLAs with each atomic service providers. In an ideal situation, the SLA negotiation could be automated; an element is required to search alternative services and negotiate the SLAs, such as mentioned in [55]. The use of SLAs in dynamic service composition could be easily applied into approaches that already enable to bind abstract services to a set of functionally similar concrete service.

3.5.4 Feedback collection system

Consumers' experiences provide "actual" insight into the services and are therefore a good information source for analyzing the softer forms of reliability. Many of trust and reputation-based approaches, especially the agent-based approaches, rely quite heavily on service final consumers, such as ratings from previous use of services and consumer feedback. The feedback can be explicit (e.g., the consumer fills out a form in consultation with the human user, such as in [41]) or implicit (the agent infers the consumer's rating based on heuristics), or the service consumer provides a rating indicating the level of satisfaction with a service after each interaction with the service (e.g., in [56]). Besides being subjective, the other drawback of the reputation-based approaches is that in the case of composite service, the feedback or rating for single services is hard to obtain, and it only gives insight into the average behavior of the service. Collecting data while the service is interacting with the consumer (e.g., in [57]) could give an objective insight into the actual reputation of the service. However, the major drawback of reputation-based approaches is that they are as such inadequate to ensure the service reliability during run-time since it cannot detect the sudden changes during service execution.

According to the literature survey in [58], the current trust and reputation systems are almost centralized where a central QoS registry is deployed to collect and store QoS data from Web service consumers [44,57,59]. However, the disadvantage of these UDDI-based approaches is that the UDDI server may become outdated in a dynamic environment where changes occur continuously. Therefore, the peer-to-peer Web services with decentralized trust and reputation techniques provide more reliable and available service systems. On the other hand, these decentralized approaches seem more complicated than centralized methods, involving a lot of communication between elements, such as in the approach described in [60] where QoS registries are organized in a P2P way to collect QoS feedback from consumers. Each registry is responsible for managing reputation for a part of service providers.

3.6 Applicability of the approach

To assist the composite service developer and provider to engineer reliable composite service, the approach/method should, above all, provide guidance for all stakeholders in their responsibilities to achieve reliability. In an ideal situation, the method should not require special user skills, but rather use of the method should be included as a part of the method user's normal activities. The financial cost and investments of introducing the method must be known, as well as the amount of work required to implement it. To be easily introduced, the method should be mature, i.e., there should be some kind of evidence about its use, applicability, benefits and costs. Mature methods are validated or used in the industry. For mature method, also tool support usually exists.

4 The results of the survey

In this section, the methods and approaches which can be applicable in developing reliable composite services as described in Fig. 1 and Sect. 2 are examined. Since the number of reliability-related work in the literature is large, we had to restrict the scope of our survey. We therefore define the criteria for the selection of approaches for this survey and concentrate only on the methods that fulfill these criteria. However, since we made an extensive literature survey, we also discuss some emerged issues revealed in the survey. We conclude the current status of the work in the literature considering the criteria and refer to some other related approaches. Some approaches were applicable only to certain framework criteria, but brought new viewpoints, thoughts or ideas to the discussion.

The selection criteria for the detailed inspected approaches were the following:

- The approach had to concentrate on composite services. The traditional software systems differ in many ways from the service-oriented systems, and therefore, development methods and approaches suitable for them are not applicable for service-oriented systems. The features such as model-driven, evaluation-based and policy-based computing [61], and run-time behavior such as service discovery, selection, composition and monitoring are typical to service-oriented systems. Composite service is here considered as a wide, distributed entity; the composite service must not be restricted to one device.
- The approach had to concentrate on reliability. The approaches concentrating "softer", non-technical reliability, such as reputation, trust or trustworthiness can also be included as long as they provide a formal way to achieve it. Approaches including reliability as a part of their QoS property can also be included as long as reliability has its own metric in analysis.
- The approach had to be applicable in service-centric context. User-centric contexts [62] promote applications that move with consumers and adapt according to changes in the available resources. Service-centric contexts [63] promote applications that allow for service adaptability, deal with service availability and support on-the-fly service composition. Context-awareness allows systems to detect changes, such as detection of a new service or device offering services with better quality, failure of a service in use or a decline in its quality and an inappropriate change in service status. Service-centric context is more convenient from the composite service provider's viewpoint, since the provider needs context-awareness for detection of new services.
- The approaches had to be applicable to any domain. As long as the domain was not restricted, we wanted to include papers considering different application areas and technologies, such as Web services and pervasive/ubiquitous systems.
- The approaches had to be applicable for composite service developer and provider. The approach had to assist the developer both in composite service engineering and in reliability engineering, considering all the phases in Fig. 1.

4.1 Context of the approaches

4.1.1 Introduction of the selected approaches

A hierarchical reliability model for service-based software system (Wang et al. [53]). The approach provides a modeling framework to analyze reliability of data, service, service pool and composition. The composition of services is modeled as a workflow of processes, and the reliability model is constructed of atomic, simple and composite processes connected by the control constructs and corresponding transition rules. The system reliability is modeled from the static and dynamic viewpoints. The static model can be used for early-stage quality prediction, before service deployment, and is generated by transforming the service process model into discrete time Markov chain (DTMC) model. The dynamic DTMC model is dynamically constructed by run-time monitoring of the service execution paths. The monitoring is used to detect changes in the system composition, configuration and operational profiles, adjusting the reliability model continuously. These are obtained from the logged operating profiles in the form of transition probabilities between services. A service pool mechanism is used for providing run-time service redundancy, maintaining a local index of the available alternative services. A prototype is provided, through use of which the reliability models can be automatically established and continuously adjusted. The approach is not restricted to any particular domain, but it requires that the services are described with OWL-S.

Context-aware dynamic service composition in ubiquitous environment (Tari et al. [64]). The purpose of the work is to enable the seamless integration of smart objects in a ubiquitous space. The approach provides a design architecture including planning algorithms and monitoring mechanism for dynamic service composition. The approach separates the concepts of abstract and concrete services. The service composition architecture consists of three types of plans: abstract template and optimal plans, and a concrete execution plan. The template plan is created using rule-based techniques, containing all possible abstract services that could compose the service. The optimal plan is created by selecting the best abstract services candidates according to their reputation and the complementarity of the parameters. Finally, the execution plan is created by selecting concrete services based on their quality of user experience (QoE) value, which is weighted according to user preferences, user's context and the environment context. The execution plan is monitored. The adaptation enables replacing the concrete services by another or even updating the optimal plan. The approach introduces a QoS-based learning mechanism, which rewards a concrete service after execution, and calculates its new quality parameters and estimates the new reputation of its abstract service accordingly.

QoS-driven run-time adaptation of service-oriented architectures (Cardellini et al. [55]). The approach provides a methodology for run-time adaptation of service systems to meet its QoS requirements in its operating environment. The approach uses two-level grammar to model the class of SOA systems managed by MOSES (MOdel-based SElf-adaptation of SOA systems) framework; the first level specifies the structure of the considered composite service, whereas the second level defines the production rules for each abstract service. The MOSES bases on the idea of binding each abstract service to a set of functionally equivalent concrete services. The MOSES framework requires as an input a set of candidate concrete services and the description of the composite service provided using a workflow orchestration language. If the description is verified belonging to the class of SOA systems, the behavioral model of the composite service is built. If a relevant change in the operating environment is detected by the monitoring activity, the model is dynamically used to calculate a re-arrangement of the available concrete services. The QoS of a composite service is calculated by recursive rules using the QoS of the concrete services, the way they are orchestrated, and the usage profile of those services. The behavioral model is used to build the template of an optimization problem, which parameters are derived from the SLAs negotiated with the composite service clients and providers, and from a monitoring activity. The adaptation policy is to select the best implementations of the composite service in a given scenario optimized in a given environment. MOSES can be applied to any composite service whose orchestration pattern matches the first level of the grammar. A prototype of the MOSES implementation is currently provided.

Reliability modeling and analysis of service-oriented architectures (Cortellessa and Grassi [65]). The approach provides a methodology for reliability modeling and analysis of SOA. The model for reliability prediction is based on a probabilistic flow graph, which is enriched with statistical information needed to support the prediction. This includes the pattern of requests addressed to other services and the information about the internal reliability of a service associated with each stage of the flow graph. Transitions from node to node of the flow graph follow the Markov property, but is extended with other kinds of control flows allowing more than one external service request to be specified within each node. The model evaluation algorithm takes the client-side perspective on reliability, assuming that the service reliability can be expressed by multiplying the probability of reaching the end state of the flow graph (calculated using Markov process) and the reliability of the network used by the client to access the service. The approach also presents an architecture that implements the methodology in SOA environment. The methodology assumes that each composite service provider publishes information concerning the service internal structure that consists of exploited external services, how the services are glued together, and how frequently they are invoked. Three different service selection policies are identified based on the published information. The methodology can be used to support the selection procedure by comparing the reliability of candidate concrete services.

A real-time reliability model for ontology-based dynamic Web service composition (Chawla et al. [10]). The main content of the approach is a feasible real-time reliability model. Reliability of a service is defined using the OWL-S profile attached to each service with two parameters; desired and marginal reliability. A service is described as a process using a process model template (PMT) into which the suitable services are searched. PMT is defined as a dynamic process model consisting of structural components, which is then instantiated into instantiated process model (IPM) by binding components of PMT into atomic or composite services. IPM extends the PMT with a set of placeholders for the details how a simple component can be bound to a selected Web service. The atomic service reliability consists of the reliability of the service and the reliability of the machine where the service is deployed. The atomic service real-time reliability is calculated using the failure intensity and execution time, and the hardware reliability using the shape and scale parameters. The parameters for the reliability calculation are stored in an OWL-S profile. The reliability of a composite service depends on its structure, the degree of independence between service components and the availability of its constitutive Web services. Reliability model for each structural component (sequence, parallel, choice and loop) is defined. The approach supports maintaining reliability at run-time by monitoring the service reliabilities in real time. However, the re-configuration of the composite service requires human interference. A prototype of service monitoring tool is provided.

Dynamic Web service selection for reliable Web service composition (Hwang et al. [66]). The approach provides a method for dynamic Web service selection for reliable Web service composition. The method is based on aggregated reliability (AR), metrics to measure the reliability of each configuration in a WS composition and two dynamic strategies that use the computed ARs for dynamically selecting atomic WSs for the composite Web service. The service composition is described using Markov chains with added states, success and failure and transition probability, and the AR of each configuration is defined recursively from the probability that the services are successfully executed in the current configuration. In AR-based selection strategy, an atomic WS is selected for each incoming operation of the composite WS so as to achieve maximum reliability. In composability and AR (CAR)-based selection strategy the ARs as well as the composabilities of configurations in selecting atomic WSs are considered. The approach takes an iterative approach to compute the vector of aggregated reliabilities considering the different possible mappings in an WS composition in order to choice of which sequence of service delegation to use. The approach can be implemented using the current WS standards; however, due to the invocation order in a set of operations, it requires using some business process composition language. A prototype has been developed using BPEL that implements the proposed approach for specifying the invocation order of a Web service.

A reliability evaluation framework on composite Web service (Ma and Chen [67]). The approach proposes a service reliability model both for atomic Web services and for composite services, and a consumer feedback-based composite service framework. The atomic service reliability bases on assumptions of independency of service reliability and is calculated using time-dependent Markov model with failure intensity and failure locating and fixing time. The approach describes the composite service structure as nodes and the relationship between nodes. Markov chain is used also to evaluate the back-up services to achieve node reliability. Finally, the aggregated composite service reliability is described as reliability of nodes and the operation relationships of subset of node set. The feedback-based framework uses feedback mechanism to collect QoS information from clients consuming the service. The collected atomic service QoS information is stored in UDDI repository, from where they are used for composite service reliability evaluation each time a change occurs in service composition. The framework can be extended to include more attributes.

QoS-aware middleware for Web services composition (Zeng et al. [40]). The approach provides QoS-aware middleware, AgFlow, for supporting quality driven Web service composition. The main features include service quality model to evaluate the quality of Web services and composite services, and two alternative service selection approaches for executing composite services. The quality model defines the QoS criteria for both elementary services and composite services. The selection process bases on the user's weight assigned to each criteria and a set of user-defined constraints. In the local optimization approach, the optimal service selection is performed for each individual task in a composite service without considering the global QoS. The QoS information of each candidate service is collected, after which a quality vector is computed for each of the candidate services. The service is selected basing on the quality vectors applying a multiple criteria decision making (MCDM) technique. In the global planning approach, QoS constraints and preferences assigned to a composite service as a whole are considered. Every possible execution plan associated with a given execution path is generated. The selection of an execution is made by relying on the MCDM technique. The simple additive weighting technique is used in both approaches whether to select the optimal service or the optimal execution plan. The approach also includes an adaptive execution engine, which reacts to changes occurring during the execution of a composite service (e.g., component services become unavailable or change their predicted QoS) by re-planning the execution. Currently, the AgFlow has been implemented as a platform that provide tools for defining service ontologies, specifying composite services using state charts, and assigning services to the tasks of a composite service. The service selection approaches can be applied to other paradigms than Web services, such as in the context of service-oriented architectures.

Toward autonomic Web services trust and selection (Maximilien and Singh [57]). The approach provides an agent-based trust framework for service selection in open environments. The framework includes a policy language to capture profiles and preferences of the service consumer and the provider, which are expressed using the concepts in the ontology. The framework enables the service selection based on the preferences of service consumers and the trustworthiness of service providers. The approach bases on software agents that are attached to each Web service, which communicate with the service consumers, calculate service quality reputation and assign a trust level to the available services. The architecture of the approach includes agency where the agents collaborate and share data collected from their interactions with the services. The approach provides detailed matching algorithms, which the agents use to select services based on the policies using semantic matching. Thus, the agents add the quality-based selection functionality between service and consumers. The agents and ontologies reside and are managed in a separate server. The services are assumed to be described using WSDL. In the current implementation, the agents are implemented in Java, but the agent may implement Web service interface to the client and thus allow cross-platform consumer-to-agent interactions.

4.1.2 Comparison of the context of approaches

The surveyed approaches are summarized in Table 1.

According to Table 1, the surveyed approach seemed to have very similar definition for composite service, except for pervasive/ubiquitous systems [64], the definition also included device services. We found papers representing different technologies; however, the offering was larger on the context of Web services. According to our survey, OWL-S seems to be the most popular format for Web service description. We found out that besides the Web services, OWL-S seems to be used in the context of pervasive and ubiquitous systems. The main content of most of the approaches was architecture or a framework that enabled services to be composed, monitored and analyzed dynamically. The work described in [40,57,66] concentrated clearly on service selection problem, and the works in [10,53,65,67] concentrated strongly on reliability analysis, providing reliability evaluation framework or models. Service composition planning was the main content in [64] and run-time adaptation in [55]. The expected benefit of each approach was the reliable or trustworthy composite service, although the approaches had different levels and methods on how to achieve it. These issues are discussed in more detailed in the next sections.

4.2 Reliability requirements and service architecture design

4.2.1 Evaluation results

Table 2 summarizes the selected approaches from the RE and architecture design viewpoint.

Of the surveyed approaches in Table 2, only [40,57] supported ontologies. The approach described in [57] suggests QoS ontology, which is used in service matchmaking by enabling service providers express quality policies and service consumers express quality preferences. The QoS ontology has three levels; the upper ontology includes the basic characteristics of all qualities and their main concepts. The middle ontology specifies domain-independent quality concepts, such as reliability, availability and security, which are then completed by a domain-specific lower ontology. The service quality model suggested in [40] consists of five quality dimensions; execution price, execution duration, reputation, reliability and availability. For each dimension, the model determines the definition of the quality element, related ser-

Table 1 Introduction of approaches

Method/approach	Composite service definition	Application area	Main content	Expected benefits
Wang et al. [53]	Composition of atomic software services	Software service based system described with OWL-S	Comprehensive system architecture, reliability modeling and analysis methods	Reliable and fault-tolerant composite service
Tari et al. [64]	Composition of intelligent atomic services (applications, sensors, robots, devices, etc)	Composite services in ubiquitous environment	Dynamic service composition architecture	Flexible and failure-tolerant service composition
Cardellini et al. [55]	Composition of network-accessible (software) services	Any software service-based systems	QoS broker architecture	Self-adaptable, dependable SOA system
Cortellessa and Grassi [65]	Assembly of software systems from preexisting components/services	Any SOA-based systems	A model for predicting and analyzing reliability in SOA framework	Reliable composite service
Chawla et al. [10]	Real-time composition of available Web services	Web services	Monitoring system architecture, reliability models	Real-time reliability evaluation of various service compositions
Hwang et al. [66]	Composite Web service orchestrated by run-time invoked Web services	Web services	Two dynamic WS selection strategies, dynamic WS selection architecture	Maximized likelihood of successful execution of composite WS operations
Ma and Chen [67]	Composition of atomic Web services	Web services	Reliability evaluation framework	Reliable and trustworthy composite Web service
Zeng et al. [40]	Composition of inter-connected Web services	Web services	Middleware platform enabling the quality driven service composition	Optimized QoS in the composite service execution
Maximilien and Singh [57]	Application composed of dynamically selected (software) services	Web services	Augmented architecture with agent framework	Dynamically configured, trustworthy application

Table 2 Reliability requirements and architecture design

Method/ approach	Reliability (QoS) ontology	Reliability requirements and means for achieve and manage reliability	Transforming reliability requirements into architecture design	Composite service architecture description
Wang et al. [53]	Not considered	Reliability requirements have not been considered. Service pool is used as fault-tolerance mechanism; not considered in requirements	Not considered	No conceptual architecture exists. Composite service is modeled as a workflow of OWL-S processes
Tari et al. [64]	Not considered	Quality requirements are specified in the consumer profile. Fault-tolerance is taken into account in the composition architecture (learning mechanism and reputation)	Not considered	Composition is described as a template plan that contains all possible abstract services that could compose the required service. Described in the form of a graph
Cardellini et al. [55]	Not considered	Involved parties state the required values for attributes in a contract (SLAs-R). Means for reliability verification are implemented in broker architecture	Not considered	Composition logic is abstractly defined as an instance of a grammar. Each abstract service is then bound to a concrete service
Cortellessa and Grassi [65]	Not considered	Bases on reliability prediction of service compositions; requirements are not considered	Not considered	Service is described as probabilistic flow graph for reliability analysis
Chawla et al. [10]	Not considered	Reliability requirements (desired and marginal reliability) are defined in the Process Model Template	Not considered	Service is described using a dynamic process template into which the suitable services are searched
Hwang et al. [66]	Not considered	Not considered	Not considered	No architecture exists. Finite state machine is used to model the permitted invocation sequences of Web service operation
Ma and Chen [67]	Not considered	Not considered	Not considered	Composite service is described as a kind of workflow process for reliability evaluation
Zeng et al. [40]	Quality model is included in service ontology	A set of user-defined constraints are expressed using a simple expression language	Not considered	A composite service is a collection of generic service tasks described in terms of service ontologies and combined according to a set of control-flow and data-flow dependencies
Maximilien and Singh [57]	QoS ontology is provided	Reliability requirements are described in the form of policies. Means for reliability verification are not discussed	Not considered	No architecture exists

vices and operations of the element, and instructions of how to compute or measure the value of the element.

The policy-based [57], SLA-based [55] and profile-based [64] approaches in Table 2 succeeded to catch the requirements in a formal way. However, it is unclear how exactly the requirements were elicited and engineered. Some approaches had their own template or profile for requirements. For the rest of the papers, it is unclear from where and in what form does the information about the required quality come from when selecting services. None of the papers considered the means to verify and manage reliability during run-time in requirements level. It is unclear how the requirements affect the architecture in the selected approaches in Table 2, since the mapping of the requirements to architecture was missing. Furthermore, it is hard to discover how the design decisions have been made since the requirements for the means for the run-time reliability achievement and management have not been defined.

The concept of "service architecture" was missing in almost all approaches surveyed, or it varied what was meant by architecture. In many cases, such as in the approaches using OWL-S, the service architecture was described when the services were already selected, and the architecture seemed to be described in the form of processes. It seems that the approaches assume that the requirements for service are known, although most of them do not define how they are achieved and mapped to conceptual services when searching suitable candidates. Instead of describing the service architecture, the approaches concentrate on service description technologies. The services were mainly described in a way that can be utilized during run-time by registering, discovering and binding services. Of the selected approaches in Table 2, only in [10,64] is the composite service modeled before service selection. In reliability model of Chawla et al. [10], a service is described using a dynamic process template. This kind of template supports the abstract service description. The framework of Tari et al. [64] uses rule-based techniques to construct a template for composition plan. The major advantage of this approach is its ability to describe abstract composite service architecture.

4.2.2 Discussion about the related literature

In our literature survey, we found that QoS ontologies have been suggested recently in several works. The dispersion among the works is large, which reveals the lack of standardization. However, it is clear that the benefits of ontologies have been widely recognized. We found only few ontologies that concentrate directly on reliability. In [68], a very simple Web Service Reliability Ontology (WSRO) is proposed, whereas the approach introduced in [69] defines and uses the reliability-metrics ontology for defining reliability requirements.

Several standards have been proposed for describing semantic Web services, such as Web Ontology Language (OWL), Web Ontology Language for Services (OWL-S), Web Service Modeling Ontology (WSMO), Web Services Description Language (WSDL-S) and DAML-S (DARPA agent markup language for services). Since the existing standards do not allow description of quality properties, several extensions have been proposed, such as [7,43,70–72]. The approaches can be considered to be applied also in the context of reliability. Several other ontologies have been suggested for service discovery and matchmaking purposes, such as [73–75]. Since there exist a multiplicity of proposed approaches for extensions of existing standards and several suggestions for QoS ontologies, it is hard to discover their differences, advantages and disadvantages. However, the diversity of approaches reveals that finding the services and binding them to a composition is hard due these stand-alone solutions. Thus, more standardization is required. The different proposals emphasize different quality attributes, and the definition problems, classification problems and representation problems still exist within quality attributes. Generally, most ontology-based approaches focus on description of static quality; they do not define dynamic quality metrics or metrics for service status variables. So far, there seem to be no formal framework to describe devices, but some proposals have been suggested, such [23,76–80]. The device ontologies enable the networked devices to discover each other's presence on the network and establish functional network services for data sharing, communications and entertainment. The disadvantage of these approaches is that currently they do not consider reliability or any other quality issues. It is still obvious that the device ontology is a necessity to describe capabilities of available devices in a standard way.

The need of new techniques and approaches for eliciting and determining service provider and customer requirements has been recognized in recent research [81] but only a few approaches have emerged. The selection of requirement elicitation techniques is always dependent on the situation at hand. Although there are some attempts to rationalize the selection of technique [82–84], there is no evidence on that one technique is better than another. More likely, the techniques complement each other, and the use of two or more techniques is always better than using only one. Formal requirements engineering methods concentrating on quality requirements exist only few. The i* framework [85] helps to detect where quality requirements originate and what kind of negotiations should take place, leading to the most appropriate architectural design decision to be used in a particular context. Extending the i* framework, the NFR (non-functional requirements) framework [86] aims to refine the quality requirements, consider different design alternatives, perform trade-off analyses and evaluate the degree to which the requirements are satisfied. NFR+ framework [87] extends the NFR Framework with measurable non-functional requirements, bridging the gap between NFRs and implementation. These kinds of frameworks help to find the reliability requirements and transfer them into the design decisions. Sindre and Opdahl [88] suggest that use of the negative form of use cases—misuse cases—can be applied in requirement elicitation. Misuse cases that embody negative scenarios and malign actors can identify and analyze threats to system operation, leading to reliability and security requirements. The DAM (Dependability Analysis and Modeling) profile of Bernardi et al. [89] assists the requirements engineers in determination of dependability requirements, focusing on reliability and availability. The approach exploits use cases/misuse cases [88] and the IEEE 830 standard for software requirements specification [90], providing an iterative workflow where the reliability and availability requirements (R & AR) elicitation and documentation are addressed within the unified process assisted by the DAM profile. The profile enables specification of R & AR in terms of quantifiable objectives or metrics and characterization of faults and failure. Based on accepted practices and worldwide standards, such as UML, the approach is easy to apply in different contexts.

In our literature survey, we found few ontology-based approaches that can be used within requirements. Xiang et al. [91] outline the Service Requirement Elicitation Mechanism (SREM), which is based on Service Requirement Modeling Ontology (SRMO) [92], utilizing its main concepts. The authors propose the Service Requirements Elicitation Process (SREP), which helps to generate a requirements model based on the concepts in ontology. The mechanism integrates the users' requirements models from different service requestors, building a requirement knowledge repository, which offers service providers requirement knowledge about the needs and preference of service requestors with regard to the service offered by them. The requirements analysis method of Kaiya and Saeki [93] is based on mapping between a requirements specification and ontological elements. For example, the requirements document is incomplete when not all elements in the ontology are related to items in the requirements specification. The quality of requirements, such as correctness and consistency, can be estimated using a defined formula after mapping requirements items onto ontological elements. Both ontological approaches enable the identification of consistent and complete requirements, and also systematic requirement management and further utilization.

We found some approaches that concentrated on bringing quality to architectural description. The approach introduced in [94] provides the means to support reliability in design following the principles of MDA [95], whereas [96] suggests a UML profile for the reliability domain. Also, the DAM (Dependability Analysis and Modeling) profile [89] is based on UML. At this moment, several UML tools enable the creation of quality profiles, e.g., Topcased (http://www.topcased.org) and Enterprise Architect (http://www.sparxsystems.com). In the quality aware software architecting and analysis approach of Ovaska et al. [97], the mapping of quality requirements with architectural elements is performed in a standard way. The approach also introduces a Quality Profile Editor (QPE) tool, which enables the user to select the appropriate metrics from the reliability ontology and set the desired value for each property. The architecture is described using annotated UML.

Only few approaches considered the architectural description of the composite service. In the approach of Ovaska et al. [97], the comprehensive architecture description is provided, having two levels of abstraction: conceptual and concrete. The conceptual level is an abstract description of architecture. Although the approach is meant for component-based software, the abstraction levels can be applied within services as well. In [9], an extension is suggested to the OWL-S ontology framework to support dynamic Web service composition. The extension allows defining the composite services at the abstract service level. Each abstract service is attached with an instance pool that includes available concrete services for the corresponding abstract service. The candidate services can be invoked from the pool dynamically during run-time.

4.3 Reliability analysis

4.3.1 Evaluation results

Table 3 summarizes the selected approaches from the reliability analysis viewpoint.

Of the selected approaches in Table 3, [53,57,65] managed to perform the reliability analysis already in service selection phase, before service binding. Almost all of the approaches enable the run-time reliability analysis; thus, the main focus seems to be on the run-time quality verification. However, approaches that perform exclusive run-time analysis cannot know the reliability of the service selection before the composite service is deployed, its execution has been started and the dynamic reliability analysis has been performed for the first time.

The different approaches to gather reliability information (defined in Sect. 3.3.2) were present in the inspected approaches (Table 3), except the use of third parties. The analyses that are based entirely on information provided by the service provider, such as in [65], cannot be considered trustworthy as such but require verification of this information. It is obvious that the dynamic quality information is required to locate some place other than the service registry due to constant updating. Attempts to enrich the UDDI registers with semantic-enriched QoS information to enable bet-

Table 3 Reliability analysis viewpoint

Method/approach	Analysis level	Data source for the analysis	Analysis technique/method	Analysis output
Wang et al. [53]	Selection phase, execution	Service providers or evaluations provided by the service brokers and consumers	Dynamically constructed DTMC model	Technical atomic and composite service reliabilities
Tari et al. [64]	Execution	Monitoring system collects information for quality parameters of concrete services	Mathematical formula (includes a learning mechanism)	Response Time, Availability and Reliability of concrete services
Cardellini et al. [55]	Execution	Execution monitor collects information about the composite service usage	Mathematical formula (recursive rules)	Average response time, cost and reliability of a given service implementation
Cortellessa and Grassi [65]	Selection phase, execution	Reliability information published by service providers	Markov processes	Reliability of atomic and composite service
Chawla et al. [10]	Execution	Data is obtained from the services using the monitoring function	Mathematical formula (recursive algorithm)	Service reliability, composite Web service reliability
Hwang et al. [66]	Execution	Transition probabilities from the transition probability matrix	Markov chains	Aggregated reliability of each configuration in a WS composition
Ma and Chen [67]	Execution	Feedback information collected from clients	Time-dependent Markov model, Markov chains, reliability SWR algorithm	Feedback-based QoS (incl. reliability) of atomic services, composite service reliability
Zeng et al. [40]	Execution	Provider advertised QoS, own system collects QoS data about services	Aggregation functions	QoS of composite services (execution price, execution duration, reputation, reliability, availability), quality vector of a composite service's execution plan
Maximilien and Singh [57]	Selection phase, execution	Provider advertised service policies. Reputation: agents collect information on their interactions with the services	Mathematical formula	Quality reputation

ter service selection still rely on static and subjective quality information. How "right" and up-to-date the quality values are cannot be verified. Furthermore, the UDDI server may become outdated in a dynamic networking environment where a service may fail or become unreachable. Solution to static information of UDDI is provided in [67], where the service QoS information stored in UDDI is updated each time a service is consumed. The trustworthiness and accuracy of the feedback information are the major challenges of the methods that are based on information obtained from earlier users of services, such as in [67]. Most of the approaches were based on building an own service monitoring system that was included as part of the service composition architecture. Therefore, the approaches ensured the trustworthy of information by acquiring it by themselves, when the information was also stored within their monitoring and composition architecture.

All of the inspected approaches had different kind of reliability analysis method. The analysis methods were mainly based on Markov chains types of models or applied mathematical formulas. Most of the analysis methods were quite mathematical, requiring many skills from the method user. Therefore, the applicability of those methods by composite service developer is quite weak, unless the guide or tool support is provided for the method. The methods were provided only as a research level; however, some of them were validated with a prototype tool. The prototypes automate some calculations of the methods and could also provide concrete proofs about the functionality of the method. The model-based analysis methods, such as [53,65,67], could be easier to be applied if the user is familiar with common Markov chain-based analysis. The approach described in [53] even enabled the dynamic construction of Markov process.

Almost all of the inspected approaches reached for both atomic and composite service reliability. The results contain mainly technical reliability. However, some approaches such as [57,67] reached for softer metrics of reliability. The approach in [67] collects feedback from consumers after they have consumed the service. In the approach described in [57], the consumers' judgments about services are aggregated into general opinions.

4.3.2 Discussion about the related literature

Although the third-party-based approaches were not involved in the selected approaches, we could made a conclusion based on our literature survey that there are multiple possibilities for third parties to assists in data gathering and service monitoring. The approach of Gouscos et al. [98] proposed that a trusted third party is tracking, updating and publishing the actual QoS of Web services. As an intermediate, Fei et al. [99] provide a usable policy-driven monitoring framework that can be used as a third-party QoS monitor for Web services. The framework includes elements for service monitoring and composite service adaption, providing users the means to define monitoring models. The monitor can work as a QoS metric value collector for a QoS registry or as a dynamic controller in service execution to adapt service execution dynamically when QoS deviation occurs. Also, the approach of Zeng et al. [100] allows users to define the QoS metrics and also to define when and how the metric values are computed. The approach introduces QoS observation metamodel, which is then used for creating observation models that then define the generic or domain-specific QoS metrics. QoS monitoring architecture includes, among others, a metric computation engine and a Web service observation manager, which provides interfaces that allow users to create observation models. The approach also includes QoS data service, which provides an interface that allows other SOA components to access QoS information.

Most of the surveyed approaches assumed that the monitoring and composition architecture also includes elements for storing of the dynamic information. Some other approaches suggested that the environment or the extended standards could store dynamic information. In the approach of Yu and Reiff-Marganiec [101], the "inContext" platform provides dynamic context information, which affects the Web service measurement at run-time; the platform dynamically stores the QoS metadata of registered services and allows for updating this at any time. However, access to the data may cost something, and the trustworthiness of the storage and the storage provider must be ensured somehow. Lee et al. [102] propose a dynamic service ranking extension to OSGi specification, suggesting that the contextual information can be attached as an attribute to service instances, and it is dynamically updated to keep up with changes in the environments. The approach proposes architecture for a service composition subsystem where the availability and quality changes of participating component services are monitored.

Most of the current quality analysis approaches are integrated with service selection or monitoring approaches and methods. The existing specifications and standards for SOA implementations have their focus in ensuring reliable message exchange between services [103,104]. The need for evaluation of service-oriented architecture has been widely recognized, but there is still a lack of common practice as to how the evaluation should be done.

4.4 Decision making logic

4.4.1 Evaluation results

Table 4 summarizes the selected approaches from the decision making logic viewpoint.

None of the inspected approaches in Table 4 used entirely the static information. Surprisingly, many approaches used the most recent data in decision making when selecting services [10,57,64,66]. These approaches can ensure the reliability of the selected services at the time of service selection. Four approaches relied entirely on the average data. Of the selected approaches, reliability was involved almost all of them as a selection criteria. Service reputation was involved in two approaches [40,64], and trustworthiness of service provider in one approach [57].

The inspected approaches had different ways of selecting services. The approach in [57] bases on a matching algorithm where the required QoS is matched with provided QoS. The different kind of matching is being made in the approach in [10] where the required service properties described in a dynamic process template are matched with OWL-S profile template associated with candidate services. The approach of Wang et al. [53] introduces a pool mechanism that can rank services according to certain criteria for selection. On the contrast, the approach of Ma and Chen [67] did not consider how the service is selected from the UDDI.

All of the detailed inspected approaches in Table 4, except [65], enabled quality adaptation. Except the approach in [10], the adaptation could occur automatically during runtime. The run-time adaptation concentrated mainly on self-configuration and self-healing; the services were replaced when there was a decrease in quality of services in use, or the services in use failed. In addition, in [55] the adaptation occurred each time, a change was detected in the operating environment. In the cases of [40,64], the adaptation included, in addition to replacement of failed or unreliable services, also the adaptation of the whole execution plan.

4.4.2 Discussion about the related literature

The non-functional-based service selection has been seen as interesting for a while, since we found plenty of approaches and several literature surveys of the area of quality-based selection approaches [12,14,18,19]. Yu and Reiff-Marganiec [12] provide first classification of non-functional-based service selection approaches, defining three dimensions to examine and differentiate approaches: policies- vs. reputation-based selection information, UDDI-extensions vs. Semantic Web Services based non-functional properties capturing, and graphic modeling vs. ontology-based modeling for capturing user preferences. Sathya et al. [14] identify criteria for quality-of-service-based service selection (QSS) approaches; QoS modeling, QoS categorization, user preferences, QoS evaluation, aggregating the evaluation of QoS, QoS properties, level of automation, coordination distribution, agent involvement and ranking algorithm. According to the surveys, the major lack of most of the approaches is the representation and modeling of QoS characteristics, metrics and inability to evaluating the QoS parameters, the QoS weightings, and fuzzy view on the QoS parameters between

Table 4 Decision making

Method/ approach	Type of data for decision making	Service selection criteria	Service selection process	Quality adaptation
Wang et al. [53]	Dynamic average data	Reliability	The pool mechanism ranks the services to ensure that the best services will be selected first	Allows the system to be automatically adjusted to the distribution and the length of service pools. In the case of failure, the system can dynamically bind to a replacement service in the pool
Tari et al. [64]	Dynamic, more importance is given to the last quality estimation	Quality of user experience and reputation of the abstract candidate services	The concrete services are selected basing on their QoE values	Execution plan is automatically adapted in case of decrease of quality of concrete services. The failed service is replaced locally by the best concrete service
Cardellini et al. [55]	Dynamic average data	Response time, cost and logarithm of reliability	An abstract service is bound to a set of concrete services; (re-)binding to different implementations to meet the QoS objectives is controlled by a coordination patterns	The model of SOA system is used to dynamically calculate a re-arrangement of the available concrete services each time a relevant change is detected in the operating environment
Cortellessa and Grassi [65]	Not considered	Not considered	Not considered	Not considered
Chawla et al. [10]	Dynamic, real-time data	Reliability	Selection is made by matching required service properties in Process Model Template with OWL-S profile template associated with each candidate services	Services can be replaced when their reliability drop to an unacceptable level, adaptation requires human inference
Hwang et al. [66]	Dynamic, recent data	Reliability	Aggregated reliability is computed for each configuration in a WS composition, and then used for selecting atomic WSs	Re-computing aggregated reliabilities if atomic services fail (unable to detect failures), selection of new services based on computation results
Ma and Chen [67]	Dynamic average data	Not considered	Services are selected informally from UDDI	If selected services fail they are substituted by backup services, unclear whether dynamically or not
Zeng et al. [40]	Dynamic, average data	Global planning: QoS constraints (execution price and duration, reputation, reliability, availability) and preferences	System computes a quality vector for each candidate service and selects services based on these vectors by applying MCDM technique	The execution plan is revised in order to optimize the QoS given a set of user requirements and a set of candidate component services if services become unavailable or there is change in their predicted QoS
Maximilien and Singh [57]	Dynamic average and recent data	Preferences of service consumers, trustworthiness of providers	Algorithms to select services based on the consumer's and service provider's policies	A service is automatically replaced at run-time if it doesn't meet the customer's needs, becomes untrustworthy, or a better service instance is found

service consumers and service providers [12,14]. Based on the related literature surveys considering the quality-based service selection, we can conclude that the issue has been inspected widely. Still, however, a standard guideline or method of how to select a reliable service is missing.

4.5 Composition and monitoring architecture

4.5.1 Evaluation results

Table 5 summarizes the composition and monitoring architecture related issues of the selected approaches.

Of the surveyed approaches in Table 5, Wang et al. [53] provide a promising means for fault tolerance; a service pool mechanism providing dynamically updated run-time service redundancy. Also, the approaches in [40,55,64,67] include some kind list or matrix of available candidate services, but it seems that they are not maintained dynamically. Thus, although the list or index of candidate services exists at the time of service selection, it is not clear what happens to the list after selection.

The surveyed approaches used monitoring activity for slightly different ways. The main purpose of most of the methods was still to detect change in service composition and quality by collecting information about the service execution and calculating the QoS values. The monitoring activity could be targeted directly to service execution or to detect changes in system composition [53]. In addition, the monitoring activity was used to monitor the interactions between the consumer and the service [57], the service usage profile [55] or the execution plan [64]. Only the work in [55] was based on SLAs. The SLA is used for specifying the conditions for service delivery, including the quality and quantity levels. The SLA Monitor collects information about reliability levels experienced by the users and offered by the service providers and notifies the adaptation manager about significant variation of service parameters.

Of the surveyed approaches in Table 5, only two approaches provide some kind of feedback system. Maximilien and Singh introduce in [57] a reputation-based approach for ontology-based dynamic service selection that is based on agent framework. The services are ranked using the quality-degree match, which is based on "what the provider advertises along with the provider's reputation for the given quality, and how the quality in question relates to other needed qualities." Maximilien and Singh [41] extend their work and propose an interacting multiagent approach that enables applications to be dynamically configured at run-time, adapting the changes in preferences of the participants. The approach takes into account that the QoS will change overtime, when the quality data must be updated regularly. The notable remark in this approach is that the recent data matter more in determining reputation. The approach of Ma and Chen [67] introduces a reliability evaluation framework that is based on collecting consumer feedback.

4.5.2 Discussion about the related literature

Other kinds of back-up plans than a list of candidates were uncovered in our literature survey. In the service selection approach of Kokash [105], several configurations with good qualities are preresolved at the service selection stage. In the case of a failure in the chosen configuration, another configuration can be chosen. The agent-based approach of Maamar et al. [63] does the concurrent composition and execution of services; the conversations with master-service-agents of the next Web services ensure that these next Web services are getting ready for execution. In dependability evaluation framework of Zheng and Lyu [106], a fault tolerance updater module automatically adjusts the fault tolerance strategy to the most proper configuration with optimal Web service candidates. The dynamic fault tolerance seems to have potential; however, currently, the QoS includes only performance. Hamadi et al. [107] provide a formal way for fault tolerance already in design time with extended Petri net model for specifying exceptional behavior of business processes. The high-level recovery policies incorporated either with a single or a set of tasks are generic directives that model exceptions at design time together with a set of primitive operations used at run-time to handle the occurrence of exceptions. However, the approaches does not discuss how the reliability of tasks, set of task or services are analyzed and how exactly the quality issues could trigger the exceptions. Friedrich et al. [108] propose a self-healing approach for service-based processes, which, on the basis of the diagnosis of the functional faults, enables that the effects of the faults on the services can be computed and used to generate a plan to repair the process. The approach provides plans for repairability both for design time and run-time, including analysis of repairability, algorithms for generating repair plans and repair execution support. However, the effects of exceptions on service reliability or any other issues relating the service reliability are not discussed; the approach concentrates only on service output monitoring during run-time with back-up plans if exceptions occurs.

We found several approaches that considered device quality and status monitoring. Some common protocols, such as UPnP [23], can be used for device advertising, discovery and also monitoring. In the UPnP-based service monitoring approach of Togias et al. [76], the UPnP devices are equipped with instances of the UPnP ontology and ontology management software. Also, the resource description model of Kaefer et al. [109] uses UPnP as a service discovery technology, consisting of a basic set of attributes including QoS parameters, device capabilities and the required resource needs. The desired objective is achieved by dynamic

Table 5 Service composition and monitoring architecture

Method/ approach	Dynamic list of candidate services	Run-time service monitoring	SLA monitoring	Feedback collection system
Wang et al. [53]	Dynamically updated index of available services	Monitoring the service execution paths	Not SLA-based	Not included
Tari et al. [64]	List of candidate services, not dynamically updated	Monitoring of the service execution plan	Not SLA-based	Not included
Cardellini et al. [55]	Pool of candidate available services, not dynamically updated	Monitor collects information about the service usage profiles	SLA monitor collects information about reliability and informs adaptation layer about violations	Not included
Cortellessa and Grassi [65]	Not included	Monitoring of the service activity is suggested (not included)	Not SLA-based	Not included
Chawla et al. [10]	Dynamic service discovery, no candidate service list exists	Service execution monitoring to detect changes in quality	Not SLA-based	Not included
Hwang et al. [66]	Not included	Not included	Not SLA-based	Not included
Ma and Chen [67]	Backup service pool exists, not dynamically updated	Not included	Not SLA-based	After client has been consumed a Web service, the feedback information is sent to UDDI
Zeng et al. [40]	Quality vectors of all candidate services are merged in a matrix. Unclear whether these are updated	Service execution monitoring to detect exceptions or changes	Not SLA-based	Not included
Maximilien and Singh [57]	Not included	Monitoring of interactions between service and consumer	Not SLA-based	Agents collect information on their interactions with the services that they selected on behalf of consumers

composition process, comprising functionality-based service lookup, resource tree generation, policy-based selection and resource-based optimization. The monitoring and analysis framework of Truong et al. [110] is based on sensors that monitor QoS and status of disparate Grid services by using a peer-to-peer Grid monitoring middleware, which stores monitoring data in distributed monitoring services. For each type of resource, e.g., machine, network path, middleware or application, the different monitoring mechanism is applied to evaluate QoS attributes. The QoS knowledge base contains analysis rules for specific metrics and resources, dependencies between monitored resources, and historical QoS data resulted from previous analyses. In the middleware-based approach of Zhang and Hansen [111], the dynamic context information is encoded in a set of self-management context ontologies. However, although some promising approaches for device monitoring already exist, they are not yet mature for reliability analysis. For example, the approach in [76] is hard to apply, since all the devices must be equipped with ontological elements. Furthermore, the approach does not consider quality issues at any level. Although the selection process in [109] takes into account the reliability and availability of the services, the recent or dynamic quality and status of the services cannot be verified. The approach relies on static information about the services, since it does not provide a method for updating the resource description model. Furthermore, it does not support dynamic re-planning if some change occurs, for example, in service execution or quality. Use of the method in [110] is very costly since each service needs a sensor to monitor it; thus, it is only suitable for small systems.

We found some other service discovery and selection approaches that were based on SLAs. The standard service-oriented architecture suggested by Janicke and Solanki [112] includes observer components that observe the interaction between services, and a policy engine that constantly evaluates the policies against the information that is provided by the observers to verify the fulfillment of QoS specified in SLA. Cardellini et al. [113] extend their work presented in [55] and introduce a brokering service that dynamically adapts at run-time the composite service configuration to fulfill the SLAs. The SLAs are negotiated with each candidate concrete service in the service pool. The approach includes WS Monitor that notifies if some services in the pool become unavailable or some relevant changes occur in the composite service environment. However, it cannot detect if new candidate services become available. Even though the new services could be discovered dynamically, the SLA negotiation still requires human inference, which restricts the dynamicity and availability of the composite service.

We found several approaches that based on user's feedback in quality analysis. The approach of Li et al. [114] uses the consumers' feedback to define both the service reputation and service provider's reputation. The approach uses intelligent agents to handle the changing environment and submit feedback to the semantic registry. The approach discusses also QoS in general, but it is hard to understand what is meant by that since it does not give any definitions or metrics. The approaches in [56,115] both extend UDDI to accommodate the QoS information and introduce a reputation manager to assign reputation scores to the services based on customer feedback and a discovery agent to facilitate the service discovery. Thangavel and Palanisamy [115] suggest a dynamic Web service discovery framework with QoS support, in which a reputation manager assigns reputation scores to the services based on customer feedback on performance. The approach uses the Certifier described in [116] to verity the quality of service for a Web service before its registration. However, the changes in quality after service registration cannot be detected. In the approach of Xu et al. [56], consumers rate the various QoS attributes of the Web services they use. These ratings are then published to provide new customers with valuable information that can be used to rank services for selection. Reputation is the only dynamic quality information, and it is entirely based on the opinions of earlier users.

Some approaches integrated the user feedback-based approaches with the quality monitoring. Zeng et al. [39] propose an extensible set of quality criteria that can be applied to all Web services. In their approach, the consumer feedback activities and monitoring activities are merged; reputation is achieved by average ranking given to the service by end users, and reliability is achieved using historical data about past invocations by calculating the number of times that the service is successfully delivered within the maximum expected timeframe. The extensible QoS model of Liu et al. [44] achieves the adequate data using the user feedback and execution monitoring. Reputation can be reached by recording the difference between the actual quality criteria and the advertised quality criteria from the service provider. Each end user is required to update QoS for the service he/she has just consumed.

4.6 Applicability of the approach

Although an enormous amount of approaches exist that consider the subject of this survey, and also several promising approaches, no accurate conclusions could be made about their maturity. In many cases, the validation of methods is based only on the authors' experiments and evaluations under laboratory circumstances, and only a prototype usually existed. It is obvious that there is a great lack of large-scale, industrial applications of these approaches.

To be applicable, the approaches require tool support for several phases. It should be carefully estimated whether the approach/method can be applied and integrated with the development method and tools of the composite service developer. Generally, all the approaches lacked tool support; however, some parts of the approaches could be supported by existing tools. For example, currently, there are several tools available that support at least the analysis that is based on Markov chains [117]. For service monitoring, some tools exist, such as Nagios (http://www.nagios.com/), which is a computer system monitor, network monitoring and infrastructure monitoring software application and Keynote system (http://www.keynote.com), which is a product family of tools for testing and monitoring of mobile content applications and services. Nagios is available as open source and used commonly in the industry, whereas Keynote system provides commercial third-party monitors. It is obvious that the different phases of the composite service design and run-time provide several opportunities for third parties. The third parties can assist in monitoring services, collecting feedback from users, analyzing reliability, providing data storages, providing trustworthy information about candidate services, ect.

The fact that the approaches surveyed exist only at the research level and that they lack tool support hinders their applicability in the industry. Since the publications covering industrial applications of the methods were missing, the cost estimation of using the approaches and methods could not be defined. It can be assumed that although the initial cost when introducing the approach is quite large, the cost of using the approach is small. However, introduction of the approaches requires a comprehensive change in composite service engineering, e.g., introducing monitoring architectures and quality verification methods. Furthermore, according to our knowledge, at this moment no public service registries exist where the candidate services can be searched. Some attempts to kind of sort out this problem exist, such as Xmethods (http://www.xmethods.com) that lists publicly available Web services and their publishers, and Rackspace Service Registry (http://www.rackspace.com) that is built in Cloud Monitoring service enabling service tracking and easy service discovery. The approaches surveyed have their own registries, or they are applicable to certain service descriptions in standard service registries, such as UDDI. More public registries, possibly maintained by third parties, are needed for service providers to advertise their services.

5 Discussion

Traditionally, a significant part of the research on achieving quality goals has been focused on software development and its internal quality attributes as in the ISO 9126-1 quality model [118]. In recent years, there has been increasing demand on verifying quality at service execution time. There exists a huge amount of papers in the literature considering quality-based service selection, monitoring of service execution and quality adaptation. This reveals that recently quality issues and quality verification have been seen as more and more crucial in development of service-oriented systems.

In our literature survey, we found many papers that matched with several criteria in our framework. However, the diversity of approaches highlights the importance of standardization, since at this moment, all these approaches represent stand-alone solutions. It is clear that more standardization is still required in several phases, such as in the definition of terms, requirement description, architecture modeling and reliability analysis, to enable fluent engineering of reliable composite services. For example, at this moment, there exist a multiplicity of proposed approaches for extensions of existing service description standards and several suggestions for QoS ontologies; still, it is hard to discover their applicability and usability, since they exist in the research level only. Despite the lack of standardization, other problems still remain that hinder the applicability of the approaches in industry.

In our literature survey, we found out that the early phases of composite service development were not supported well in the selected approaches. QoS ontologies were rarely used. Reliability requirements and means for achieve and manage reliability were not formally defined, and therefore, also the influence of these requirements to architecture design decision could not be detected. The service architecture was described only in few approaches in a conceptual or abstract level that supported service selection. However, all the selected approaches succeeded to perform reliability analysis. This was quite confusing, since the main purpose of the analysis is to verify that the requirements are being met, and in several cases, the requirements were missing. All approaches performed dynamic analysis, and some approaches managed to perform the analysis even when selecting services. The common problems with many approaches were that the analysis method lacks tool support, or the method cannot be used as such but require special analysis models or familiarity with mathematical models. As being mathematical and lacking tool support, the reliability analysis methods currently require much special skills and knowledge from the method user. Also, the effort for user is great as these mathematical methods may be difficult to integrate with current tools. However, the model-based analysis methods could enable the automatic transformation from service architectural description to analysis models.

Almost, all the selected approaches based their decision making on dynamic information, which enabled them to verify the actual reliability of the service. The current reliability, i.e., the reliability at the time of selection, was resolved in several approaches. The approach had different selection processes for atomic services. Almost, all of them supported quality adaptation, mainly in a form of replacing the failed or unreliable services. Some of the approaches enabled automatic adaptation. Most of the surveyed approaches succeeded to reach and manage reliability each in their own way. The monitoring architecture as such can be understood as a means for fault tolerance. Almost, all of the selected approaches included service monitoring function. A list or pool of candidate services existed in several approaches. Only one of the selected approaches was SLA based; the SLA monitoring was included only in this approach. The user feedback collection mechanism was included only in two approaches that were aiming at trustworthiness.

All of the approaches would be hard to apply at this moment. First of all, they denote a big change in the whole composite service RE and design processes, requiring the design of a composition and monitoring architecture and reliability analysis methods and models. The lack of tool support complicates the introduction of the approaches. Finally, the approaches are very small-scaled and restricted; some standards may be referred but the implementation is based on stand-alone solutions. The diversity of different approaches concerning service selection and monitoring, and quality adaptation reveals that the concept of how to conduct these issues is not mature. Therefore, more applied research is needed to create standard methodology and related tools for industry use.

In addition to standardization, some kind of new actors are needed, possible third parties, which facilitate the work of composite service developers by rationalizing the required activities. First of all, these third parties could maintain the public registries of available services from where everybody could search services. Reliable composite service design and execution opens other opportunities for third parties; they could provide monitoring support, information storages and analysis support. This means that the composite service developer is not obligated to implement all the issues discussed in Sect. 3. However, when using third parties, the cost and trustworthiness issues emerge as an important.

6 Conclusions

Recently, there has been an increasing demand for quality-based service selection and verification of quality at service execution time. A large amount of research papers in the literature exist that considers service selection based on QoS, dynamic service composition, monitoring of service execution and dynamic quality adaptation. The purpose of our work was to detect the current status of the research literature aiming to reliable composite service engineering. We first defined and described the different phases and elements in reliable composite service modeling and execution. These phases are described from the composite service developer and provider viewpoints when developing and managing reliable composite service. Then, we developed and presented an evaluation framework with the criteria derived from these phases. We performed a literature survey to discover the status of research fields of the criteria. The papers that covered the criteria of our framework best were selected for further examination, and the way of how they consider the criteria was evaluated. In addition, we also discussed the status of the work in the literature and other related approaches that are applicable to certain subset of framework criteria bringing new viewpoints, thoughts or ideas.

The main purpose of our framework is to assist in revealing how the criteria are taken into account in the current approaches in the research literature. Thus, it exposes the potential and shortcomings of the approaches considering each of the criteria. The criteria of the framework are purposely high level to be applicable to different context, being therefore domain and implementation independent. The composite service developer requires means, methods and techniques for each phase of engineering reliable services. The criteria of our framework can be easily refined to more detailed level, if taken into a certain domain and technological context. Thus, the framework can also be thought as a domain and implementation independent "tool" for developers for evaluating different approaches for their own cases. The developer can utilize the framework for detecting the potential of approaches and for creating a collection of methods and a tool chain supporting the methods (or parts of them), to be applicable in his/hers domain and technological context.

Our literature survey revealed that none of the approaches fully cover all the criteria of our framework. We found out that several promising approaches exist that claim to enable reliable, fault-tolerant or self-adaptable composite service, but they still have some shortcomings. Special attention should be paid to the early phases of composite service development, since those were supported the weakest. There still exist no agreed customs of how to define reliability and reliability requirements, how to transform these requirements to design and how to verify whether they are met. Generally, the approaches surveyed did not agree on what information is required for service selection, how to achieve this information and how the services should be described to attain reliability. The service monitoring and quality adaptation, however, were supported quite well, and thus, most of the approaches succeeded to reach reliability at some level. However, the diversity of approaches highlights the importance of standardization in several phases in composite service design and run-time. Several issues restrict the applicability of the surveyed approaches in common use at this moment. The approaches are often isolated and also not compatible with the existing standards and practices. Most of all, they lack tool support. The required changes in the composite service RE and design phases also hinder the applicability of the approaches in industrial use. None of the surveyed approaches could provide any validation or proof about their maturity. However, the amount of research in the area of reliable service engineering reveals that common interest in the topic is high at the moment. It is obvious that the development of standard methodology and tools is inevitable to encourage industry to change their service engineering methods and practices for engineering reliable composite services.

References

1. OASIS (2008) Reference architecture for service oriented architecture 1.0. OASIS SOA reference model. https://www.oasis-open.org/
2. Erl T (2007) SOA principles of service design. Prentice Hall, Englewood Cliffs, NJ
3. Lyu MR (1996) Handbook of software reliability engineering. Mcgraw-Hill, New York, NY
4. Liu F, Tong J, Mao J, Bohn R, Messina J, Badger L, et al. (2011) NIST cloud computing reference architecture, recommendations of the National Institute of Standards and Technology, vol 500, p 292. NIST Special Publication. National Institute of Standards and Technology, Gaithersburg, MD
5. Barry DK (2003) Web services and service-oriented architectures: the savvy manager's guide. Morgan Kaufmann Publishers, San Francisco, CA
6. Parliamentary Office of Science and Technology (2006) Pervasive computing (postnote). No. 263. Parliamentary Office of Science and Technology, London
7. Thomson G, Bianco S, Mokhtar SB, Georgantas N, Issarny V (2008) Amigo aware services. In: Constructing ambient intelligence, part 7, communications in computer and information science, 1, vol 11, pp 385–390
8. Mokhtar SB, Georgantas N, Issarny V (2007) Cocoa: conversation-based service composition in pervasive computing environments with QoS support. J Syst Softw 12:1941–1955
9. Dong J, Sun Y, Yang S (2006) OWL-S ontology framework extension for dynamic web service composition. In: Proceedings of the eighteenth international conference on software engineering & knowledge, engineering (SEKE'2006), pp 544–549
10. Chawla H, Xu H, Zhou M (2011) A real-time reliability model for ontology-based dynamic web service composition. In: 23rd international conference on software engineering & knowledge engineering (SEKE'2011), pp 153–158. Miami Beach
11. Zhang L (2008) EIC editorial: introduction to the body of knowledge areas of services computing. IEEE Trans Serv Comput 1(2):62–74
12. Yu HQ, Reiff-Marganiec S (2008) Non-functional property-based service selection: a survey and classification of approaches. Non-Funct Prop Serv Lev Agreem Serv Oriented Comput Workshop 411:13–25
13. Papazoglou M, Traverso P, Dustdar S, Leymann F (2008) Service-oriented computing: a research roadmap. Int J Coop Inf Syst 17(2):223–255
14. Sathya M, Swarnamugi M, Dhavachelvan P, Sureshkumar G (2011) Evaluation of QoS based web- service selection techniques for service composition. Int J Softw Eng (IJSE) 1(5):73–90
15. Rao J, Su X (2004) A survey of automated web service composition methods. First international workshop on semantic web services and web process composition. SWSWPC, San Diego, California, pp 43–54
16. Dustdar S, Schreiner W (2005) A survey on web services composition. Int J Web Grid Serv 1(1):1–20
17. Alamri A, Eid M, El Saddik A (2006) Classification of the state-of-the-art dynamic web services composition techniques. Int J Web Grid Serv 2(2):148–166
18. Urbieta A, Barrutieta G, Parra J, Uribarren A (2008) A survey of dynamic service composition approaches for ambient systems. In: Proceedings of the 2008 Ambi-Sys workshop on software organisation and MonITored of ambient systems (SOMITAS '08). (1), 8 p
19. Stavropoulos TG, Vrakas D, Vlahavas I (2011) A survey of service composition in ambient intelligence environments. Artif Intel Rev, pp. 1–24
20. Ibrahim N, Le Mouël F (2009) A survey on service composition middleware in pervasive environments. Int J Comput Sci Issues (IJCSI) 1:1–12
21. Satyanarayanan M (2001) Pervasive computing: vision and challenges. IEEE Pers Commun 8(4):10–17
22. Weiser M (1991) The computer for the 21st century. Sci Am 256(3):94–104
23. ISO/IEC (2008) ISO/IEC 29341–1:2008–UPnP device architecture–part 1: UPnP device architecture version 1.0
24. (2003) OSGi service platform–release 3. IOS Press, Amsterdam
25. ISO (2008) ISO 9001:2008 quality management systems–requirements. International Organization for Standardization. http://www.iso.org/
26. Papazoglou M, Traverso P, Dustdar S, Leymann F (2007) Service-oriented computing: state of the art and research challenges. IEEE Comput 40(11):38–45
27. Liu L, Yu ESK, Mei H (2009) Guest editorial: special section on requirements engineering for services - challenges and practices. IEEE Trans Serv Comput 2(4):318–319
28. Bucchiarone A, Cappiello C, Nitto ED, Kazhamiakin R, Mazza V, Pistore M (2009) Design for adaptation of service-based applications: main issues and requirements. In: Proceedings of ICSOC/ServiceWave workshops, pp 467–476
29. OMG (2003) Unified modeling language (UML) 2.0 specification. Object Management Group. http://www.omg.org/spec/ML/2.0/
30. OMG (2003) UML profile for schedulability, performance, and time specification. Object Management Group. http://www.omg.org/spec/SPTP/
31. Aagedal JO, de Miguel MA, Fafournoux E, Lund MS, Stolen K (2004) UML profile for modeling quality of service and fault tolerance characteristics and mechanisms, technical report 2004–06-01. Object Management Group
32. Reussner RH, Schmidt HW, Poernomo IH (2003) Reliability prediction for component-based software architectures. J Syst Softw 66(3):241–252
33. Grassi V (2005) Architecture-based dependability prediction for service-oriented computing. In: Architecting dependable systems III, p 299. Springer, Berlin
34. Dai YS, Xie M, Poh KL, Liu GQ (2003) A study of service reliability and availability for distributed systems. Reliab Eng Syst Saf 79(1):103–112
35. Goseva-Popstojanova K, Mathur AP, Trivedi KS (2001) Comparison of architecture-based software reliability models. In: Proceedings of the 12th IEEE international symposium on software reliability engineering (ISSRE 2001), pp 22–31
36. Immonen A, Niemelä E (2008) Survey of reliability and availability prediction methods from the viewpoint of software architecture. Softw Syst Model 7(1):49–65
37. Lee K, Jeon J, Lee W, Jeong S-H (2003) QoS for web services: requirements and possible approaches. W3C notes
38. Zhou T, Zheng X, Song WW, Du X, Chen D (2008) Policy-based web service selection in context sensitive environment. In: IEEE congress on services 2008–part I, pp 255–260
39. Zeng L, Benatallah B, Dumas M, Kalagnanam J, Sheng QZ (2003) Quality driven web services composition. Proceedings of the 12th

international conference on World Wide Web (WWW). Budapest, Hungary, pp 411–421
40. Zeng L, Benatallah B, Ngu A, Dumas M, Kalagnanam J, Chang H (2004) QoS-aware middleware for web services composition. IEEE Trans Softw Eng 30(5):311–327
41. Maximilien EM, Singh M (2004) A framework and ontology for dynamic web services selection. IEEE Internet Comput 8(5):84–93
42. Mani A, Nagarajan A (2002) Understanding quality of service for web services. IBM white paper
43. Garcia D, Toledo M (2006) Semantics-enriched QoS policies for web service interactions. WebMedia 06:35–44
44. Liu Y, Ngu AHH, Zeng L (2004) QoS computation and policing in dynamic web service selection. In: Proceedings of the 13th international conference on World Wide Web (WWW), pp 66–73
45. Amoroso E, Watson J, Marietta M, Weiss J (1994) A process-oriented methodology for assessing and improving software trustworthiness. In: Proceedings of the 2nd ACM conference on computer and communications, security, pp 39–50
46. Jøsang A, Ismail R, Boyd C (2007) A survey of trust and reputation systems for online service provision. Decis Support Syst 43(2):618–644
47. Horn P (2001) Autonomic computing: IBM's perspective on the state of information technology. Technical report. IBM Corporation, NY, USA
48. Avizienis A, Laprie J, Randell B, Landwehr C (2004) Basic concepts and taxonomy of dependable and secure computing. IEEE Trans Dependable Secur Comput 1(1):11–33
49. Salehie M, Tahvildari L (2009) Self-adaptive software: landscape and research challenges. ACM Trans Auton Adapt Syst 4(2): 1–42
50. Yau SS, An HG (2011) Software engineering meets services and cloud computing. Computer 44(10):47–53
51. OASIS (2005) Web services distributed management (WSDM). http://www.oasis-open.org/committees/wsdm
52. Delgado N, Gates AQ, Roach S (2004) A taxonomy and catalog of runtime software-fault monitoring tools. IEEE Trans Softw Eng 30(12):859–872
53. Wang L, Bai X, Zhou L, Chen Y (2009) A hierarchical reliability model of service-based software system. In: 33rd annual IEEE international computer software and applications conference, vol 1, pp 199–208
54. Keller A, Ludwig H (2003) The WSLA framework: specifying and monitoring service level agreements for web services. J Netw Syst Manag 11(1):57–81
55. Cardellini V, Casalicchio E, Grassi V, Lo Presti F, Mirandola R (2009) QoS-driven runtime adaptation of service oriented architectures. ESEC/SIGSOFT FSE, pp 131–140
56. Xu Z, Martin P, Powley W, Zulkernine F (2007) Reputation-enhanced QoS-based web services discovery. In: IEEE international conference on web services (ICWS), pp 249–256
57. Maximilien EM, Singh MP (2004) Toward autonomic web services trust and selection. In: Proceedings of the 2nd international conference on service oriented, computing, pp 212–221
58. Wang Y, Vassileva J (2007) Toward trust and reputation based web service selection: a survey. Trans Syst Sci Appl 3(2):118–132
59. Manikrao US, Prabhakar TV (2005) Dynamic selection of web services with recommendation system. In: International conference on next generation web services practices, p 117. Seoul, Korea
60. Vu L, Hauswirth M, Aberer K (2005) QoS-based service selection and ranking with trust and reputation management. In: Meersman R, Tari Z (eds) CoopIS/DOA/ODBASE 2005 LNCS, vol 3761. Springer, Heidelberg, pp 466–483
61. Tsai WT, Jin Z, Wang P, Wu B (2007) Requirement engineering in service-oriented system engineering. In: Proceedings of the IEEE international conference on e-business engineering (ICEBE '07), pp 661–668
62. Roman M, Campbell RH (2002) A user-centric, resource-aware, context-sensitive, multi-device application framework for ubiquitous computing environments. No. UIUCDCSR-2002-2282
63. Maamar Z, Mostefaoui SK, Yahyaoui H (2005) Toward an agent-based and context-oriented approach for web services composition. IEEE Trans Knowl Data Eng 17(5):686–697
64. Tari K, Amirat Y, Chibani A, Yachir A, Mellouk A (2010) Context-aware dynamic service composition in ubiquitous environment. In: IEEE international conference on communications (ICC), pp 1–6. Cape Town
65. Cortellessa V, Grassi V (2007) Reliability modeling and analysis of service-oriented architectures. In: Test and analysis of web services, pp 339–362
66. Hwang S, Lim E, Lee C, Chen C (2008) Dynamic web service selection for reliable web service composition. IEEE Trans Serv Comput 1(2):104–116
67. Ma J, Chen H (2008) A reliability evaluation framework on composite web service. In: IEEE international symposium on service-oriented system, engineering, pp 123–128
68. Wang X, Li B, Liao L, Xie C (2011) Ontology-based reliability evaluation for web service. In: IEEE 35th annual computer software and applications conference (COMPSAC), pp 348–349
69. Zhou J, Ovaska E, Evesti A, Immonen A (2011) OntoArch reliability-aware software architecture design and experience. In: Dogru A, Bicer V (eds) Modern software engineering concepts and practices: advanced approaches. IGI Global, USA, pp 48–74
70. Wang X, Vitvar T, Kerrigan M, Toma I (2006) A QoS-aware selection model for semantic web services. Service-oriented computing ICSOC 2006:390–401
71. Zhou C, Chia L, Lee B (2004) DAML-QoS ontology for web services. In: Proceedings of the IEEE international conference on web services, pp. 472–479
72. Chaari S, Badr Y, Biennier F (2008) Enhancing web service selection by QoS-based ontology and WS-policy. In: 23rd ACM symposium on applied computing (SAC 2008). Ceará, pp 2426–2431
73. Galizia S, Gugliotta A, Domingue J (2007) A trust based methodology for web service selection. In: Proceedings of international conference on semantic, computing, pp 193–200
74. Dobson G, Lock R, Sommerville I (2005) QoSOnt: a QoS ontology for service-centric systems. In: EUROMICRO-SEAA, pp 80–87
75. Yu C, Junyi S, Yang Y, Zhizhong L (2009) A QoS ontology for semantic service discovery. In: International conference on networking and digital society, pp 108–111
76. Togias K, Goumopoulos C, Kameas A (2010) Ontology-based representation of UPnP devices and services for dynamic contextaware ubiquitous computing applications. In: Third international conference on communication theory, reliability, and quality of service, pp 220–225
77. Bandara A, Payne T, de Roure D, Clemo G (2004) An ontological framework for semantic description of devices. In: International semantic web conference (ISWC). Hiroshima, 2 p
78. W3C (2004) Composite capability/preference profiles (CC/PP): structure and vocabularies 1.0. http://www.w3.org/TR/2004/REC-CCPP-struct-vocab-20040115/
79. Foundation for Intelligent Physical Agents (2002) FIPA device ontology. Foundation for Intelligent Physical Agents, Geneva, Switzerland. http://www.fipa.org/
80. Driscoll D, Mensch A (2009) Devices profile for web services (DPWS).OASIS standard. https://www.oasis-open.org/standards
81. Lichtenstein S, Nguyen L, Hunter A (2005) Issues in IT service-oriented requirements engineering. Australas J Inf Syst 13(1):176–191

82. Zhang Z (2007) Effective requirements development–a comparison of requirements elicitation techniques. Software quality management XV: software quality in the knowledge society.British Computer Society, pp 225–240
83. Nuseibeh B, Easterbrook S (2000) Requirements engineering: a roadmap. In: Future of software engineering, ICSE '00, pp 37–46
84. Maiden N, Rugg G (1996) ACRE: selecting methods for requirements acquisition. Softw Eng J 11(3):183–192
85. Chung L, Gross D, Yu E (1999), Architectural design to meet stakeholders requirements. In: Proceedings of the TC2 first working IFIP conference on software, architecture (WICSA1), pp 545–564
86. Chung L, Nixon B, Yu E, Mylopoulos J (2000) Non-functional requirements in software engineering. Kluwer Academic Publishers, Boston, Dordrecht
87. Yrjönen A, Merilinna J (2009) Extending the NFR framework with measurable non-functional requirements. In: Proceedings of the 2nd international workshop on non-functional system properties in domain specific modeling languages, NFPinDSML2009. Denver, Colorado
88. Sindre G, Opdahl AL (2000) Eliciting security requirements by misuse cases. In: Proceedings of the 37th international conference on technology of object-oriented languages and systems, TOOLS-Pacific 2000, pp. 120–131
89. Bernardi S, Merseguer J, Lutz RR (2010) Reliability and availability requirements engineering within the unified process using a dependability analysis and modeling profile. In: European dependable computing conference, pp 95–104
90. IEEE (1998) IEEE std 830–1998, IEEE recommended practice for software requirements specifications. IEEE, New York
91. Xiang J, Liu L, Qiao W, Yang J (2007) SREM: a service requirements elicitation mechanism based on ontology. In: 31st annual international computer software and applications conference, COMPSAC 2007, pp 196–203
92. Liu L, Chi C, Jin Z, Yu E (2006) Strategic capability modelling of services. In: The 2nd workshop of service-oriented computing consequences and experience of requirements (SOCCER 2006). Paris
93. Kaiya H, Saeki M (2005) Ontology based requirements analysis: lightweight semantic processing approach. In: Fifth international conference on quality software (QSIC'05), pp 223–230. Melbourne, Australia
94. Rodrigues GN, Roberts G, Emmerich W, Skene J (2003) Reliability support for the model driven architecture. In: Proceedings of the 2nd IEEE-ACM-SIGSaFT ICSE workshop on software architectures for dependable systems (WADS"03), pp 79–98
95. Miller J, Mukerji J (2003) MDA guide version 1.0.1. Object Management Group. http://www.omg.org/mda/
96. Cortellessa V, Pompei A (2004) Towards a UML profile for QoS: a contribution in the reliability domain. in: Proceedings of the fourth international workshop on software and performance, pp 197–206
97. Ovaska E, Evesti A, Henttonen K, Palviainen M, Aho P (2010) Knowledge based quality-driven architecture design and evaluation. Inf Softw Technol 52(6):577–601
98. Gouscos D, Kalikakis M, Georgiadis P (2003) An approach to modeling web service QoS and provision price. In: Fourth international conference on web information systems engineering workshops (WISEW'03), pp 1–10. Italy, Rome
99. Fei L, Fangchun Y, Kai S, Sen S (2008) A policy-driven distributed framework for monitoring quality of web services. IEEE international conference on web services (ICWS 2008), pp 708–715. Beijing, China
100. Zeng L, Lei H, Chang H (2007) Monitoring the QoS for web services. In: Service-oriented computing–ICSOC 2007, pp 132–144
101. Yu HQ, Reiff-Marganiec S (2008) A method for automated web service selection. In: IEEE congress on services–part I, pp 513–520
102. Lee WLC, Ko S, Lee S, Helal A (2007) Context-aware service composition for mobile network environments. In: 4th international conference on ubiquitous intelligence and computing (UIC2007), pp 941–952
103. Iwaza K, Durand J, Rutt T, Peel M, Kunisetty S, Bunting D (2004) Web services reliable messaging WS-reliability 1.1. OASIS standard. https://www.oasis-open.org/standards
104. Fremantle P, Patil S (2009) Web services reliable messaging WS-ReliableMessaging 1.2. OASIS standard. https://www.oasis-open.org/standards
105. Kokash N (2006) A service selection model to improve composition reliability. In: International workshop on artificial intelligence for service composition (AISC), pp 9–14. Riva del Garda, Italy
106. Zheng Z, Lyu MR (2009) A runtime dependability evaluation framework for fault tolerant web services. In: Workshop on proactive failure avoidance, recovery and maintenance (PFARM 2009) at the 39th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp A9–A14. Estoril, Portugal
107. Hamadi R, Benatallah B, Medjahed B (2008) Self-adapting recovery nets for policy-driven exception handling in business processes. Distrib Parallel Databases 23(1):1–44
108. Friedrich G, Fugini M, Mussi E, Pernici B, Tagni G (2010) Exception handling for repair in service-based processes. IEEE Trans Softw Eng 36(2):198–215
109. Kaefer G, Schmid R, Prochart G, Weiss R (2006) Framework for dynamic resource-constrained service composition for mobile ad hoc networks. In: UBICOMP, workshop on system support for ubiquitous computing
110. Truong HL, Samborski R, Fahringer T (2006) Towards a framework for monitoring and analyzing QoS metrics of grid services. In: 2nd IEEE international conference on e-science and grid computing, p 65. Amsterdam, Netherlands
111. Zhang W, Hansen KM (2008) SemanticWeb based self-management for a pervasive service middleware. In: Second IEEE international conference on self-adaptive and self-organizing systems, SASO '08, pp 245–254
112. Janicke H, Solanki M (2007) Policy-driven service discovery. In: Proceedings of the 2nd european young researchers workshop on service oriented, computing, pp 52–62
113. Cardellini v, Casalicchio E, Grassi V, Lo Presti F (2010) Adaptive management of composite services under percentile-based service level agreements. In: 8th international conference on service oriented computing (ICSOC, 2010), pp 381–395. California, San Francisco
114. Li J, Ma D, Han J, Long X (2009) Toward trustworthy semantic web service discovery and selection. In: Proceedings of the 6th international conference on autonomic and trusted computing, ATC 2009, pp 209–220. Brisbane
115. Thangavel R, Palanisamy B (2009) Efficient approach towards an agent-based dynamic web service discovery framework with QoS support. In: International symposium on computing, communication, and control (ISCCC), pp 74–78
116. Ran S (2003) A model for web services discovery with QoS. SIGecom Exch 4(1):1–10
117. Fugua NB (2003) The applicability of markov analysis methods to reliability, maintainability, and safety. Reliab Anal Cent START Sheet 10:8
118. ISO/IEC (2001) ISO/IEC 9126–1 international standard: software engineering–product quality. Part 1: quality model. International Organization for Standardization. http://www.iso.org/