SOCA (2012) 6:231–248
DOI 10.1007/s11761-012-0105-3

SPECIAL ISSUE PAPER

CRP: context-based reputation propagation in services composition

Shiting Wen · Qing Li · Lihua Yue · An Liu · Chaogang Tang · Farong Zhong

Received: 30 March 2011 / Revised: 22 October 2011 / Accepted: 1 November 2011 / Published online: 6 May 2012

Abstract For a number of services with similar functionality, reputation has been regarded as one of the most important methods to identify good ones from bad ones. However, a composite service, which is composed of multiple component services, obtains only one score (or feedback) after every invocation. In order to compute the reputation of each component service, it is necessary for the composite service to distribute this score to its component services. How to achieve a fair distribution is a challenging issue, as each component service may perform differently in contributing to the success or failure of the composite service. Although several efforts have been made for this problem, they do not consider the context of composition, which makes the distribution unfair. Therefore, in this paper, we propose a fair score distribution framework which combines the context of component services and their runtime performance. We distinguish two aspects contexts of a component service: structure-related importance and community-related replaceability, and adopt graph theory and dominating relationship technique to compute them, respectively. Experimental results show that our approach can achieve a more reasonable and fair score distribution than other existing methods.

Keywords Web services · Reputation · Distribution

S. Wen (✉) · L. Yue · A. Liu · C. Tang
Department of Computer Science and Technology,
University of Science and Technology of China, Hefei, China
e-mail: wst1029@mail.ustc.edu.cn

L. Yue
e-mail: llyue@ustc.edu.cn

A. Liu
e-mail: liuan@ustc.edu

C. Tang
e-mail: tcg@mail.ustc.edu.cn

S. Wen · Q. Li · L. Yue · A. Liu · C. Tang
Joint Research Lab of Excellence, CityU-USTC Advanced
Research Institute, Suzhou, China

Q. Li
e-mail: itqli@cityu.edu.hk

S. Wen · Q. Li · A. Liu · C. Tang
Department of Computer Science,
City University of Hong Kong, Hong Kong, China

A. Liu
State Key Laboratory of Software Engineering,
Wuhan University, Wuhan, China

F. Zhong
Department of Computer Science,
Zhejiang Normal University, Jinhua, Zhejiang, China
e-mail: zfr@zjnu.cn

1 Introduction

Web services are an emerging and promising technology in distributed applications. A web service, identified by a URI, is a software application whose interface and bindings are capable of being defined, described, and discovered by XML artifacts, and support direct interactions with other software applications using XML-based messages via Internet-based protocols. Web service are self-contained, modular business process applications that are based on the industry standards, such as WSDL, (to describe), UDDI (to publish and discovery), and SOAP (to communicate) [1]. In this new computing paradigm, the functionality granularity of services should be limited from a reusability point of view [2]. Thus, multiple services need to be composed into a new value-added one, resulting in a composite service, to fulfill users' complex requirements. Nowadays, this can be implemented in a quite flexible and efficient way through WS-BPEL [3], which has been approved as an OASIS standard in 2007.

In web service environment, many services with identical functionality but different non-functional properties are provided by different providers. The provider usually makes a promise of quality about the provided service but may fail partially or fully to deliver this promise (the provider may exaggerate the capability of the provided service by making a good promise, which is difficult to meet, for the sake of attracting more requests), which will bring down the quality of the whole composite service. Recently, there are many trust and reputation models for web services, which have been proposed to assist users to avoid interacting with un-honest services, such as ([4–13]). However, only a few solutions of them have considered that a service can play two roles, where a service can either be invoked independently by a terminal user (in the reminder of this paper, the word of "user" expresses the same meaning as the word of "terminal user") or be invoked indirectly by a composite service. When building a service's reputation we should consider both roles of the service seriously, since a service will obtain a score by different ways due to their under different roles. For example, if a service is invoked by a user, it will obtain an independent score about the satisfaction of this invocation obviously. However, if a service is invocated indirectly by a composite service, it only obtains a distributing score from the composite service, rather than directly from a user. As for the first case (a service is invocated by a user), there have been many existing methods which can be applied in our model [4,7]. As for the second case (a service is indirectly invocated by a composite service), the service reputation we try to build should consider the fairness of the distributing method(s). In this paper, however, we only focus on the second case, since it is a challenging issue.

To the best of our knowledge, only two works [12,14] so far focused on distributing the score of a composite service to its component services. The method in [14] distributes a composite service's score only in terms of each component service's deviation between runtime performance and its advertised value of QoS, which does not consider the context of a service composition to reflect which component services should be more important than others in contributing to the composite service successfully or unsuccessfully. Even though the authors consider the context of composition by assigning an importance value to each component service, this technique in [12] is relatively simple, and besides it is hard for the user-assigned value to reflect the actual importance of a component service. In this paper, we propose a distribution framework which combines context and runtime performance of component services to distribute composite service's score to its component services to ensure that our distribution technique is fairer than other existing works. In our framework, the context includes "structure-related importance" and "community-related replaceability", and both of them will be defined in Sect. 5. The main contribution of this paper is threefold:

- We propose a context-based model, which consists of structure-related importance and community-related replaceability, as the basis of distributing composite service's feedback (score) to its component services.
- The distribution framework which combines the context information (importance) and runtime performance of the component service is used for distributing composite service's score to its component services. It can ensure the fairness of our distributing model.
- The experiments are conducted to evaluate the effect of component services, which have different structure-related importance or community-related replaceability value, to the performance of whole composite service. A hybrid distributing method is also conducted to verify that our framework can achieve fair reputation propagation.

The rest of the paper is organized as follows. Section 2 discusses some related works. Section 3 introduces some preliminaries and a motivating example to be used throughout the reminder of the paper. Section 4 presents a reputation model in web service composition. Section 5 gives the details of our distribution mechanism. Section 6 describes the experiments to evaluate our approach. Finally, Sect. 7 concludes the paper and sheds light on future research.

2 Related works

In this section, we review some related works on reputation and trust mechanism and approaches. Trust and reputation management are important for evaluating unknown parties. The trust and reputation research do not limit to a single field. Various of disciplines including economics, computer science, marketing, politics, sociology, and psychology have studied reputation in several aspects [15]. Recently, trust and reputation research has gotten a great momentum in both theories and applications in computer science field. In the theoretical areas, reputation has been studied by game theory, Bayesian network, social network and hidden Markov models and so on. In the application areas, trust and reputation management systems have being used in e-commences, p2p networks, grid networks, multi-agent systems, file-sharing systems and so on. It is impossible for us to review all representative works here due to the limitation of space. In the following, we only focus on some literatures which are related to our work. A good overview of reputation mechanism in other disciplines can be found in [4].

To our best knowledge, [8] is the first piece of works on web service reputation. The authors propose an approach to

compute service reputation dynamically according to QoS history and user preference for QoS parameters. Reputation management prototype systems can be found in [9,10], where the authors proposed a reputation management infrastructure for composite web services. In [11], the authors proposed a trust management method to assess services reputation, but this method is quite simple because only the number of positive feedbacks is compared to the total number of feedbacks over a period of time.

A number of research works focus on establishing the credibility of the rating with an assumption that the feedback may be un-honestly. In [4] and [16], the authors introduce a reputation assessment framework for trust establishment among web services. The majority rating is proposed to assess whether the feedback is honest or not. The main shortcoming of this work lies in that the factor of majority rating is not always effective since some minority feedbacks may actually be the true state of current execution. And multiple malicious users may create a colluding to cheat for obtaining a high credibility.

Trust issues in composite services have been tackled in [12,14]. In [12], the authors introduce an approach which enables a composite service to distribute the reputation value to its sub-component services. Two main factors are considered in their method: (1) the importance of each component service in a composite service; (2) the past experiences of component services. However, the importance in this work is assigned by users. In [14], the authors distribute the composite service' utility deviation value to its component services depending only on their performance, which is limited since they do not consider the fact that each component service may have different contributions to the composite service's success or failure. A method of using a vector to represent the reputation of web service was presented in [13], where the authors use three dimensions to represent the service trust. However, all of those three dimensions are only obtained by services' history behaviors. In [11], the authors proposed different scoring functions to calculate the reputation based on the feedback data for customized trust evaluation; three reputation scoring functions (eBay, PeerTrust, and Exponentially Weighted Moving Average) have been designed for their prototype system deployable in both LAN and WAN. Unlike existing works on web service reputation evaluations, our work derives services' scores according to their context information and the deviation of services between its actual QoS values and advertised values in every invocation.

Reputation based systems have also been widely used in social agent systems, peer-to-peer systems, and grid computing system. Regret [17] is a reputation system that adopts a sociological approach for computing reputation in a multi-agent societies in an e-commerce environment. Regret system incorporates social network analysis in different points of the model. It employs both individual and social components of social evaluations where the social dimension refers to the reputation inherited by individuals form the groups they belong to. The system has a hierarchical ontology structure that allows to consider several types of reputation at the same time. The combination of complementary methods that use different aspects of the interaction and social relations, allows the agent to calculate reputation values at different stages according to its knowledge of the society. However, the proposed scheme requires a minimum number of interactions to make correct evaluations of reputation. A certified Reputation (CR) [18] allows agent to actively provide third-party references about previous performance as a mean of building up the trust in terms of their potential interaction partners. It can quickly establish trust with little cost to involve parties. However, some inherited shortcomings still exist, such as, unreliable information providing by third-parities.

In PeerTrust [7] model, the authors propose a framework to quantify and compare the trustworthiness of peers. It is argued that peer uses similarity measures to weigh opinions of those peers that have provided similar ratings for a common set of past partners. The EigenTrust [19] is proposed to compute peers reputation based on power iteration. The system computes each peer's global trust value in terms of a matrix of normalized local trust values, which takes into consideration the entire system's history with each single peer. The main idea of this work is based on the notion of transitive trust that a peer will has a high perception of peer who have provided honest interaction with it since the peers had honest it is more likely for the peers with honest interactions to report their local trust value accurately. In [20], the authors present a method through dynamically selecting a few power nodes that are most reputable by using a distributed ranking mechanism to construct a robust and scalable trust modeling scheme. They use a Distributed Hash Table (DHT) to implement the distributed ranking mechanism which is the same as [7,19]. In [21], the authors present a reputation evaluation technique based on Bayesian leaning technique. In their work, the first-hand information is exchanged frequently and the second-hand information is merged, if it is compatible with the current reputation rating.

In [22], two types of reputation algorithms based on discrete and Bayesian evaluation of ratings have been presented to compute trust in a large distributed system. At the same time, the discrete combination and the fuzzy logic combination are also proposed to combine direct trust and reputation. In [23], the trust system uses reply consistent of values to predict honest. It assumes that each peer has a set of trusted allies, and obtains them by asking one or more trusted allies to send recommendation request for the target peer to the recommender. The source peer would compare the recommendation it gets directly with the one received by the trusted

allies. However, this method can not detect dishonest peers that provide consistent nt replies.

In [24–26], the game theory-based techniques were applied for online reputation systems. Most of the game theoretic models assume that stage game outcome is observable. However, the feedback based reputation mechanisms rely on private and subjective ratings of stage game outcomes. Thus they need an incentive for submitting feedback and giving a truthfulness of the rating for its interactive partners after each transaction.

3 Preliminaries

In this section, we introduce web service model and its interaction model as a foundation of our work and use an application scenario to motivate our work. The application scenario illustrates the motivation of our work, where we distribute the composite service's score in a fair manner to reflect different contribution of each component service to the composite service's execution quality.

3.1 Web service model

Typical interactions on the service include four entities: web services, service providers, service registries and service consumers. A web service is self-contained, modular business process applications that based on the industry standards and supports direct interaction with other software applications using XML-messages via internet-based protocols. A service provider is an entity that provides the services (i.e., makes it available to users). Meanwhile, a service can be provided to one or more publicly known places. A service registry is a place where services can be registered in it for easily being discovered by users. The registry manages all of registered services such as records services' advertised QoS capabilities.

We present several definitions below as the basis of our model for building the reputation of web services.

Definition 1 (*Service*) A service j is a software module which can fulfill a specific functionality with a set of non-functionality attributes (i.e., QoS: Quality of Service), which is denoted as vector Q_j.

The QoS attributes have different taxonomies from different viewpoints. In [27], the authors distinguish two types of QoS attributes: positive and negative. For a positive QoS attribute, a higher value means a higher quality. For a negative QoS attribute, a higher value means a lower quality. For example, availability and reliability are positive while response time and price are negative. In [28], the authors distinguish three types of QoS from an aggregation point of view: additive, multiplicative, and aggregated by Min-operator. For an additive QoS attribute, the QoS of a composite service is a sum of the QoS of its component services. Response time and price are two examples. For simplicity, we consider only negative and additive QoS attributes in this paper, but our work can be easily extended to other types QoS attributes by adopting the techniques introduced in [27,28]. Meanwhile, multiple services, which can fulfill the same functionality, may form a community for attracting more requests or competing to others. Then we have the simple community definition as following;

Definition 2 (*Community*) A service community C can be seen as a collection of web services with a common functionality although these web services have distinct non-functional properties (QoS) [14].

We use function $C(j)$ to get the community which the service j belongs to. Note that a community contains not only atomic services but also composite ones because both of them expose only an interface described by WSDL files. More details about community will be discussed in the next section.

It is common to attach a utility function to a service to represent its value to users. Note that, this utility function may be the financial gain from the completing services, or simply a private utility value, as commonly used in decision theory. For example, the authors propose in [29] the following function for web services:

$$u(t)=\begin{cases} u_{\max} & \text{if } t \le t_{\max} \\ u_{\max}-\delta(t-t_{\max}) & \text{if } t_{\max}<t<t_{\max}+u_{\max}/\delta \\ 0 & \text{if } t \ge t_{\max}+u_{\max}/\delta \end{cases} \quad (1)$$

Here, $u_{\max} \ge 0, t_{\max} \ge 0 \text{and} \delta > 0$, According to this function, a service earns a maximum utility $u_{\max}$ if it completes within a given deadline $t_{\max}$. If the service is delayed, the utility decays linearly with a slope of δ until it equals 0.

The above function works well when the utility only depends on the response time of services. In other words, it offers a reasonable utility to a service with only one QoS attribute. However, a service usually has multiple QoS parameters, each of which makes a different contribution to the whole utility. To aggregate these contributions, we first need to scale the utility in Eq. (2) as follows:

$$u_s = \frac{u(t)}{u_{\max}} \quad (2)$$

Based on the scaled utility of every QoS parameter, we propose the following utility function:

Definition 3 (*Utility function*) The utility function that a user k attaches to a service j is a weighted mean of scaled utilities

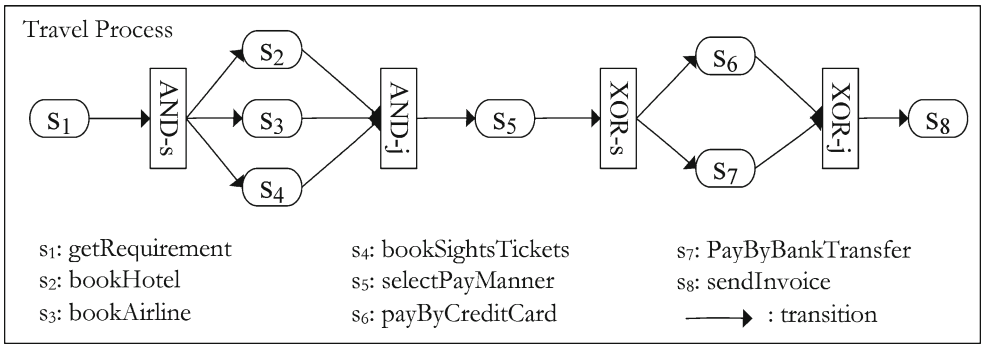


Fig. 1 Example of a composite service

coming from $|Q_j|$ QoS parameters:

$$u_{k,j} = \sum_{g=1}^{|Q_j|} w_g \times u_s(Q_{j,g}) \tag{3}$$

Here, $w_g (0 \leq w_g \leq 1, \sum_{g=1}^{|Q_j|} w_g = 1)$ is the weight attached to the utility $u_s(Q_{j,g})$ coming from the gth QoS parameter. Note that these weights are assigned by user according to their personal preferences to indicate the user's concerns on each QoS dimension.

A user k can assess the expected utility of a service j according to the utility function and the expected average QoS of the service. Formally, we have:

$$E(u_{k,j}) = \sum_{g=1}^{|Q_j|} w_g \times u_s(Q_{j,g}^{av}) \tag{4}$$

Here, $Q_{j,g}^{av}$ is the average expected value of $Q_{j,g}$.

3.2 Application scenario

We present an example to explain why a composite service needs to distribute its score to its component services and how to distribute it so as to ensure the fairness. In this example, we consider a composite service, namely travel process. We use this example to explain the key concepts throughout the paper.

The composite service is shown in Fig. 1. First, the user k invokes the composite service Travel Process with a set of initial inputs, such as the name or id of the user, the destination, and the departure and return date. Next, the composite service invokes a component service (service s_1) to receive all of those input parameters. Then it concurrently invokes booking hotel, booking airline and booking sights tickets services. After reserving replies of flight tickets, hotel rooms and sights tickets, the travel process selects the way to pay for those tickets' fees. Here there are two ways for the user to pay the fees, paying either by a credit card or by bank transfer. Finally the travel process invokes the invoice service to send a receipt to the user.

Table 1 Execution time of component services

Service name	Exe. time (ms)	Service name	Exe. time (ms)
s_1	200	s_5	200
s_2	700	s_6	500
s_3	200	s_7	500
s_4	300	s_8	300

Figure 1 describes an execution process of the travel process. In service-oriented paradigms, web services run on the Internet which is a dynamic and open environment. Thus travel process is susceptible to a wide variety of failures. In the service runtime, some component services may violate their advertised QoS quality. For the sake of simplicity, we use execution time as an example to illustrate our concern. Suppose that each component service has an expected average execution time when it is published. As shown in Table 1 we list all component services' expected average execution time.

Assume that the user selects the credit card to pay the fee, and requires that the execution time of the travel process should be replied within 1,900 ms. However, assume the services s_2 and s_3 don't complete within 700 and 200 ms, respectively and both of them are delayed 200 ms, we can clearly see that the travel process can not complete within 1,900 ms. After clear observation, we find that if service s_2 can complete in 700 ms, the travel process can still fulfill their deadline constraint reply within 1,900 ms even though the service s_3 is delayed 200 ms. However, if the service s_3 completes within 200 ms, the travel process also violates their deadline constraint while the service s_2 delays 200 ms. We can obviously conclude that the component services with different contributions to composite service performance even though they may have the identical deviation of performance. In other case, if s_3 and s_5 are all delayed 200 ms, we can also observe that the travel process can not complete with 1,900 ms. Even though s_3 and s_5 have the same execution

Table 2 Variables and their meanings

Notations	Meanings
$k/j/\text{cs}$	A user k/a component service j/a composite service cs
Q_j	A vector of QoS parameters of service j
$\text{Score}_{k/\text{cs},j}$	A score of a service j obtains from user k/cs after each invocation
R_j	Indicates the reputation of service j
d/d_R	Degree centrality score/ extended degree centrality score
Imp_j	Structure-related importance of service j
DisCtx_j	Distribution context of service j which combines Imp_j and Rep_j
$P(EP_m)$	Execution probability of mth execution plan

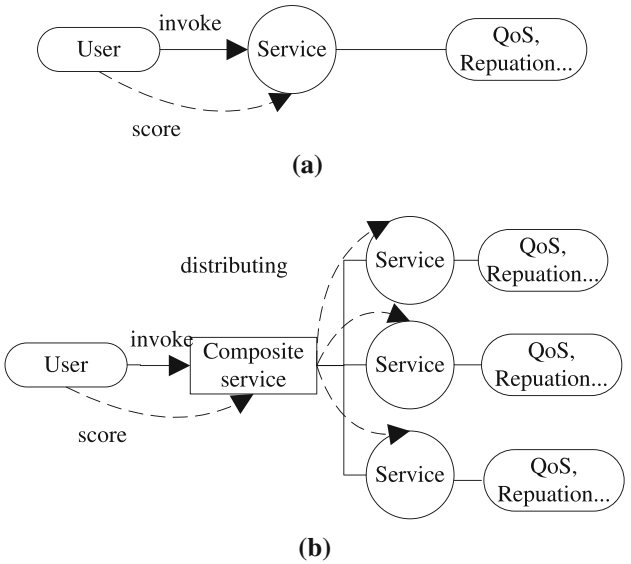


Fig. 2 Web services usage scenarios. **a** Invoke a simple service, **b** invoke a composite service

time (200 ms), however, if s_5 can complete in 200 ms, the whole composite service can meet its deadline constraint, but if s_5 completes in 200 ms, the travel process still can not meet its deadline.

After finishing the invocation, the user will give only an feedback to the composite service (Travel Process) even though the functionality of the composite service is fulfilled by eight component services. The reason is that the user can only see the front-end composite service, and s/he is unware of the back-end component services. Thus, it is still necessary to distribute composite service's overall feedback to every component service. However, how to distribute is a challenge issue. From this case, we can easily see that even though the services with the same expected QoS quality and the identical delayed value, they may still have the different contributions to the whole composite service's performance since they have different locations in the composite service.

4 Reputation model

In this section, we describe a model of reputation of web services to provide a formal basis for our work. We further propose a distribution framework which can be used to illustrate the process of distributing the score of a composite service to its component services. A unique characteristic of our distribution framework is that we have considered the context of component services in a composite service.

First, we list in Table 2 the variables to be used in subsequent sections.

According to Cambridge Online Dictionary [30], reputation is an opinion that people in general have about someone or something based on past behavior or character. Thus, web service reputation systems depend on their past behaviors to evaluate the services' and their interacted partners' reputation. There are two service invocation patterns: simple service invocation pattern and composite service invocation pattern [31].

In the web service paradigm, a service can be a simple service (the functionality is fulfilled by itself) or a composite service (the functionality is fulfilled by other services). However, user can not distinguish these two types of services, since the services only provide interface for invocation and their implementation details are transparent to the users. Figure 2 shows these two types of service invocations. The first case is rather simple: a user selects a simple service (not containing any sub-service) for invocation, and then it obtains a feedback (score) about this invocation. After receiving the feedback from the user, the service does not need to distribute the feedback to any other sub-services (actually, no sub-services exist). In the second case, a user selects and invokes a composite service, which aggregates several component services to fulfill its requirement. After receiving a feedback (score) from the user per each invocation, the composite service needs to fairly propagate feedback (score) of composite service to all of its component services.

Then, we consider a single user's opinion of whether it is a simple service invocation pattern or a composite service invocation pattern. A service j is usually invoked by a user k on demand, and then after finishing (regardless of successfully or un-successfully) the invocation v, the service j will obtain a score about how satisfactory on this invocation v. Specifically, if a service performs better than what it has promised, it will bring more value or score to the user. Then we have the following $Score$ definition.

Definition 4 (*Score*) When a service completes, its actual runtime QoS value may deviate from the expected average QoS value, so it may be different between the actual utility and the expected average utility. Then, the score of a service equals to the utility difference in invocation, as shown in the following:

$$\text{Score}_{k,j} = u_{k,j} - E(u_{k,j}) \tag{5}$$

It is a real number between -1 and 1. If actual utility is larger than the expected utility, the user k is said to gain the utility difference ($\text{Score}_{k,j}$ has a positive value); otherwise, the user k is said to lose the utility difference($\text{Score}_{k,j}$ has a negative value).

Considering the fact that the user may invoke one service multiple times, we have the following definition of general score.

Definition 5 (*General score*) If a service j has been invoked m times by a user k, then the general score that a user k has towards the service j is a weighted mean of scores in the m invocations:

$$\text{GerScore}_{k,j} = \frac{\sum_{v=1}^{m} \text{Score}_{k,j}^{v} \times \lambda^{m-v}}{\sum_{v=1}^{m} \lambda^{m-v}} \tag{6}$$

Here, λ ($\lambda \in (0, 1)$) is the weight attached to the user k's score of service j in the invocation v (i.e., $\text{Score}_{k,j}^{v}$). In this paper, Principle 1 (given below) is employed to determine a weighted mean of scores in the m invocations, which is adopted by many existing works such as [13,31].

Principle 1 The reputation value of a service is calculated according to the service's past experience performance in a recent period, and higher weights should be assigned to later data points.

If other users want to invoke this service, they first evaluate its reputation in terms of the general score which has been obtained by many pre-invoked users. Some existing prevalent methods calculate the reputation of service j by a summation of all of general scores which j obtains [c.f., Eq. (7)].

$$R_j = \sum_{k \in L} \frac{\text{GerScore}_{k,j}}{|L|} \tag{7}$$

Here, $L \neq \Phi$ is a set of users who have interacted with service j in the past (if $L = \Phi$, it indicates that the service j is a newcomer service that needs to be evaluated by some bootstrapping techniques).

Meanwhile, a composite service, which is composed of multiple component services, will also obtain a single score after each invocation:

$$\text{Score}_{k,\text{cs}} = u_{k,\text{cs}} - E(u_{k,\text{cs}}) \tag{8}$$

Formula (8) has the same meaning as Formula (5). And cs indicates the composite service, of which composition detail is opaque. Thus, after each invocation, the composite service cs can only obtains a single score only. However, the functionality of the composite service is implemented by all of its component services. Therefore, the composite service should distribute all of component services a score. We have the following equation:

$$\text{Score}_{k,\text{cs}} = \sum_{j=1}^{|\text{cs}|} \text{Score}_{\text{cs},j} \tag{9}$$

Here, $\text{Score}_{k,\text{cs}}^{v}$ is the score which the composite service propagates to the component service j after an invocation. And |cs| is number of component services in the composite service cs. Since the component services may have different contributions to the composite service's performance, it is a challenge issue to distribute the score of composite service to its component services.

In the next section, we focus on how to distribute the composite service' score ($\text{Score}_{k,\text{cs}}$) to all of its component services and each of them will obtain a score ($\text{Score}_{\text{cs},j}$). We propose a context-based technique to ensure the distributing model more effective and fair than other existing approaches.

5 Distribution framework

In this section, we present a framework to distribute a composite service's score to its component services after each invocation. The framework shown in Fig. 3 contains four major modules. The first module parses the composite service's structure-related information and calculates each component service's centrality score. The second module captures each component services' community-related context at the phase of each task binding to a concrete service. The third module records the component service and its component service's runtime information such as runtime QoS performance of each component services. The last module implements our distribution function to propagate the score of the composite service to each individual component service.

Based on the discussion above, the composite service is responsible for distributing its utility score to all of its component services after each invocation. In our framework, we consider two factors when the composite service distributes its score to its component services. One factor is the context (consisting of structure-related importance and community-related replaceability) of a component service in the composite service. The other factor is the runtime performance of component services. The context of a component service, which is the focus of this paper, includes two facets:

Definition 6 (*Structure-related importance*) The structure-related importance indicates a component service's structure context information in a composite service.

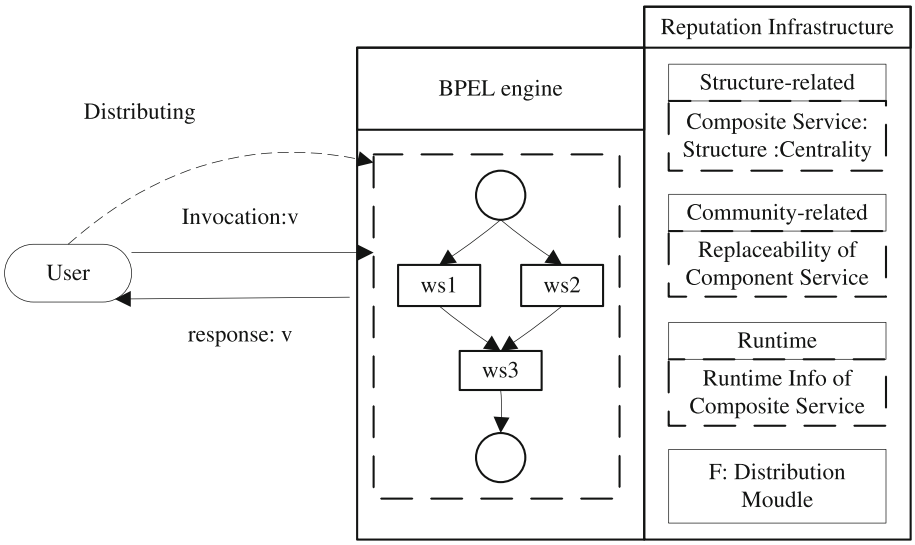


Fig. 3 Architecture of reputation propagation

It can reflect the contribution of component services to a composite service's success or failure execution, whose value can be calculated according to the "centrality" score in a graph, which is a mature theory to evaluate a node's importance in graph.

Definition 7 (*Community-related replaceability*) The community-related replaceability indicates a component service's replaceable context information in a service community which has the same functionality.

It can reflect the replaceable level of a component service if it fail in a composite service execution, whose value can be calculated according to the dominating relationship in a "community". The dominating relationship technique also provides an intuitive way for evaluating the significance of an object relative to others in a dataset, i.e., how many other services it can "dominate (c.f. definition 9)" in a community. Then we will detail how to distribute the score of a composite service to its component services by a series of steps.

5.1 Structure-related importance

A composite service (through aggregating multiple other atomic and composite services) can be seen as a workflow (our distributing model solves composite service recursively until to atomic service in terms of the requirement), which interacts with each other according to a process model.

A workflow is composed of states and transitions. In the proposed composition framework, the transitions of a workflow are labeled with events, conditions and assignment operations over process variables. State can be basic or compound. Basic states are labeled with invocations to web services operations. Compound states contain one or several sub-workflows.

A simplified workflow specifies a composite service Travel Process, which is illustrated in Sect. 2 (Application Scenario), is depicted in Fig. 1. Following our previous work [32], we use workflow patterns to describe the dependencies among component services. Figure 4 shows six types of workflow patterns considered in our current study: sequence, AND-split, AND-join, loop, XOR-split, and XOR-join. Specifically, AND-split means $s_2, \ldots, s_n$ could be executed in parallel after s_1 finishes execution. AND-join means s_1 can be executed only after all its predecessors $s_2, \ldots, s_n$ finish execution. XOR-split means only one of $s_2, \ldots, s_n$ could be executed after s_1 finishes execution. XOR-join means s_1 can be executed only after the activated service (among $s_2, \ldots, s_n$) finishes its execution.

Since the composite service may contain XOR-split and XOR-join patterns; the composite service may have multiple alternative execution plans. We have the following definition.

Definition 8 (*Execution plan*) An execution plan represents a sequence of services to complete a composite service execution.

Figure 1 gives an example of a composite service travel process's execution plans. In that example, since the travel process has one XOR-split pattern after service s_5, there are two execution plans, called EP_1 and EP_2, respectively, as shown in Fig. 5. In the execution plan EP_1, service s_6 is executed after s_5, while in execution plan EP_2, service s_7 is executed after service s_5.

For each execution plan, it can be represented as a directed graph. And each component service and transition is represented as a node and an edge, respectively. In the graph theory, centrality denotes the importance of a node in a graph after computing specific scores [33]. Four types of centrality metrics, namely degree, closeness, between ness and eigenvector, are widely accepted. Computing the score of centrality

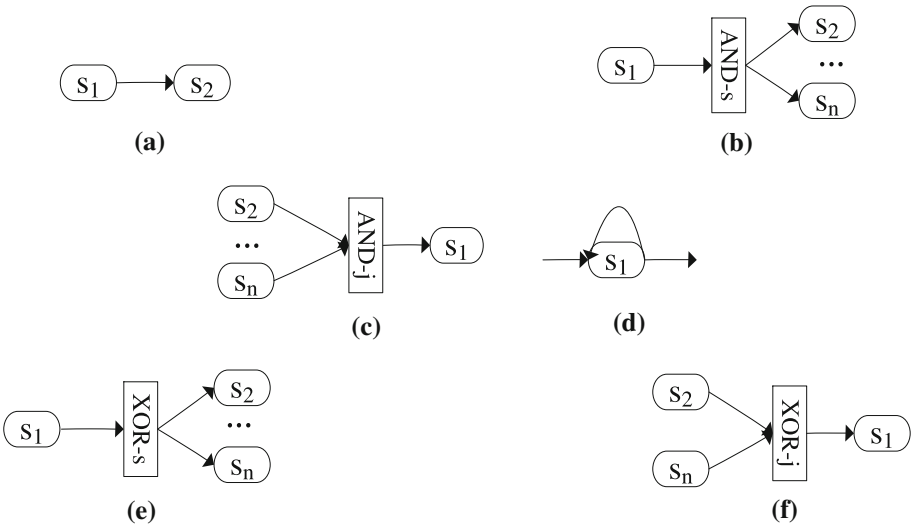


Fig. 4 Workflow patterns. **a** Sequence, **b** AND-split, **c** AND-join, **d** loop, **e** XOR-split, **f** XOR-join

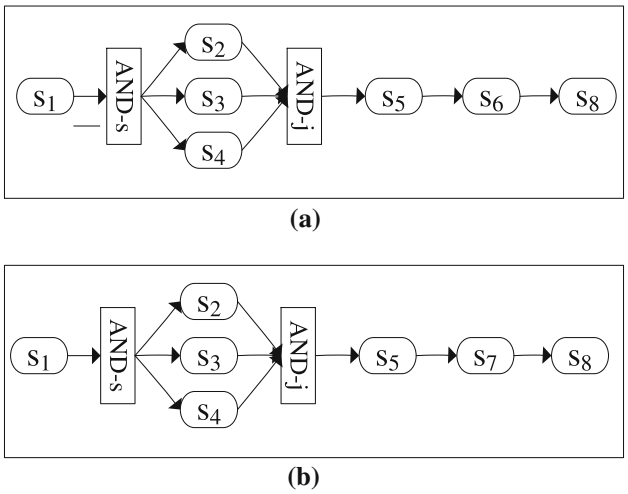


Fig. 5 Two execution plans of travel process. **a** Execution plan EP_1, **b** execution plan EP_1

of a node considers the number of edges (incoming and outgoing) connecting to this node to other nodes in the graph. The higher the score is, the higher the importance is. In this work, we only use degree centrality metric to calculate the centrality score of a node to represent its structure-related importance.

The degree centrality of a node is defined as the number of links that the node is connected to, and measures the involvement of the node in the graph. As for each execution plan, it is represented as a directed graph in which each component service is represented as a node, we have the following in-degree and out-degree formalized expressions:

$$d_{\text{in}}(j) = \frac{\sum_{s=1,s\neq j}^{N} x_{s,j}}{N-1} \tag{10}$$

$$d_{\text{out}}(j) = \frac{\sum_{t=1,t\neq j}^{N} x_{j,t}}{N-1} \tag{11}$$

Here j represents a component service; s and t represent all other incoming and outgoing component services of j, respectively. $N(>1)$ is the total number of component services, and x is the adjacency matrix, in which the cell $x_{s,j}/x_{j,t}$ is defined as 1 if service s/j is connected to service j/t, and 0 otherwise. The degree centrality analysis shows the importance of a component service in a composite service.

The degree centrality has been generally extended to the sum of weights when analyzing weighted graphs. Here, the labeled weight is the connected service's reputation which indicates the strength of service. This measure has been formalized as follows:

$$d^R_{\text{in}}(j) = \frac{\sum_{s=1,s\neq j}^{N} R_s \times x_{s,j}}{\sum_{s=1,s\neq j}^{N} R_s} \tag{12}$$

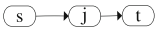
$$d^R_{\text{out}}(j) = \frac{\sum_{t=1,t\neq j}^{N} x_{j,t} \times R_t}{\sum_{t=1,t\neq j}^{N} R_t} \tag{13}$$

Here, R_s/R_t is the reputation of service s/t. The extended weighted degree is equal to the definition of degree if the graph is binary, i.e., each service has a reputation of 1. Conversely, in weighted graph, the outcomes of these two measures are different. Since service strength takes into consideration the reputation of connected services, which has been the preferred measure for analyzing weighted graphs. However, a service's strength is a blunt measure as it only takes into consideration a service's total level of involvement in the graph, instead of the number of other services to which it connects.

Claim 1 *The importance of a component service is influenced by the reputation value of its connected component services.*

Proof Consider a simple composite service cs_0, which contains three component services s, j and t, as shown in Fig. 6,

Fig. 6 A composite service (cs_0)



for example, if both s and t have higher reputation than j, the performance of cs_0 can be largely determined by j. Instinctively, we think that service j has higher importance to determine the performance of composite service cs_0 in this situation. Conversely, even though the service j performs well, the composite service cs_0 may still perform badly if both services s and t have lower reputation than j. In this case, we think that service j has low importance to determine the performance of cs_0 in this situation. □

In addition, it is clear from Formula (12) and (13) that the higher reputations s and t have, the more importance j has. Meanwhile, if s and t only one has a higher reputation and the other one has a low reputation, we clear see that the service j's in-degree/out-degree importance is high or low, respectively.

The similar analysis can be conducted when component services j has multiple incoming services and outgoing services. We omit the detailed proofs here due to space limitation.

Since degree and strength can be both indicators of the level of involvement of a node in the surrounding graph, it is important to incorporate both measures when calculating the centrality (importance) of a node.

In an attempt to combine both degree and strength, we use a tuning parameter, which determines the relative importance of the number of links compared to link's weights. We formally propose the following measure:

$$d_{\text{in}}^{R\alpha}(j) = d_{\text{in}} \times \left(\frac{d_{\text{in}}^{R}}{d_{\text{in}}}\right)^{\alpha} = d_{\text{in}}^{(1-\alpha)} \times (d_{\text{in}}^{R})^{\alpha} \tag{14}$$

$$d_{\text{out}}^{R\alpha}(j) = d_{\text{out}} \times \left(\frac{d_{\text{out}}^{R}}{d_{\text{out}}}\right)^{\alpha} = d_{\text{out}}^{(1-\alpha)} \times (d_{\text{out}}^{R})^{\alpha} \tag{15}$$

Here, α is a positive tuning parameter that can be set according to users' preference. If it is set below 0.5, then a high degree is favorable; otherwise, a low degree is favorable.

We can aggregate in-degree and out-degree centrality to obtain the centrality for each service. As shown in following:

$$d^{R\alpha}(j) = \omega \times d_{\text{in}}^{R\alpha}(j) + (2-\omega) \times d_{\text{out}}^{R\alpha}(j) \tag{16}$$

As for the execution plan, the start service and end service only have the out-degree and in-degree, respectively. Thus, we set ω equal 0 and 2 for the first service and end service respectively. ω for middle services will be set between 0 and 2. Here, for the sake of simplicity, we set ω to 1. This setting implies that we have the same concern on the in-degree and out-degree importance.

The structure-related importance of a component service is calculated according to its centrality score [c.f. Eq. (16)]. We have a more formal definition in the following phase.

We have multiple execution plans of a composite service if it contains XOR-split patterns. The probability of execution of each execution plan is different. We can easily assess each execution plan's probability of execution from the composite service past execution experience. For example, Fig. 3 shows that there are two execution plans in travel process, and we assume that the composite service has been executed 100 times in a past period including that EP_1 has been executed 80 times and EP_2 has been executed 20 times. We can obviously obtain that the probability that EP_1 executes is 0.8(80/100) and the probability that EP_2 executes is 0.2(20/100).

Each component service may locate in one or multiple execution plan(s). We calculate the structure-related importance of a component service based on discussed-above centrality score. Given a component service j, $EP(j)$ indicates the set of execution plan(s) in which service j locates in a composite service cs. The structure-related importance of service j can be calculated as follows:

$$\text{Imp}_{j,\text{cs}} = \sum_{m=1}^{|EP(j)|} d^{R\alpha}(j) \times P(EP_m) \tag{17}$$

$$P(EP_m) = \frac{N(EP_m)}{N(\text{cs})} \tag{18}$$

Here, $P(EP_m)$ indicates the probability of mth execution plan. The $P(EP_m)$ can be calculated according to formula (18), where $N(EP_m)$ is the number of EP_m which is executed and $N(\text{cs})$ is the number of cs which is executed in the latest past time window.

5.2 Community-related importance

A service community (c.f. definition 2) C can be seen as a collection of web service with a common functionality although these web services have distinct non-functional properties such as different QoS qualities [34]. A composite service sends a request to a community C to invoke one of its members. Then C will check all of its members and select the member service j, which contents the requirement of request to fulfill the request. Therefore, the more the services are, which can fulfill a concrete request, the higher the replaceability of each is. We can use the number of services which j dominates in the community C to indicate j's Community-related replaceability. Then, the dominating relationship is given by the following definition.

Definition 9 (*Dominating relationship*) Consider a service community C, and two service $s, j \in C$, Q represents a set of QoS parameters of this service community C. *s dominate j*, denote as $s \prec j$, iff s is as good or better than j in all QoS parameters in Q and better in at least one parameter in Q. (i.e., $\forall g \in [1, |Q|]$, $Q_{j,g} \leq Q_{s,g}$, and $\exists g \in [1, |Q|]$, $Q_{j,g} < Q_{s,g}$). Here, we assume that all of QoS parameters are negative.

Namely, smaller value is preferable to larger ones at all dimensions. Further, if a service $j \prec s$, and $s \prec t$, then $j \prec t$. We use notation $\prec$ to indicate the dominating relationships between services.

According to definition 9, we can easily define the dominating score of service j, as follows:

$$\mathrm{dom}(j) = |\{s \in C | j \prec s\}| \tag{19}$$

Here, C is the community which service j and s belongs to, Namely, j and s located in the same community. The $\mathrm{dom}(j)$ is the number of services dominated by service j, and the following property holds for function dom:

$$\forall j, s \in C, j \prec s \Rightarrow \mathrm{dom}(j) > \mathrm{dom}(s) \tag{20}$$

Therefore, we can define a nature ordering of the services in the same community, based on function of dom.

We can use the following formula (21) to calculate service j's community-related replaceability in view of the composite service cs, given by:

$$\mathrm{Rep}_{j,C} = \frac{|C| - \mathrm{dom}(j)}{|C|} \tag{21}$$

Here, $|C|$ denotes the number of services in the community C, and $\mathrm{dom}(j)$ indicates the dominating score of the service j.

We can easily observe that if a service has higher dominating score, the expected service performance is better than other service in the identical community. If this service fails, we can hardly replace it just another service without degrading the whole composite service's QoS constraints. Otherwise, if a service with a lower dominating score in a community, we can easily replace it when its fails. Note that, we just discuss the replaceability of a service here not considering how to bind a workflow task with a selected service.

The meaning of Eq. (21) is that there are multiple services in the community and service j's community-related replaceability is the ratio of the number of un-dominated services with the total number of services of community C. When service j fails, the number of services which can replace j without degrading the quality of the composite service is an importance factor which will influence whether the composite service can successfully complete or not. For example, there are two service s and t in the same community. If s dominates t, it indicates that s is better than t, which can surely reflect the importance of a component service. This is because the more services which are dominated by t, the larger dominating score it will be. However, once service t fails, fewer other services can replace it without bringing down the composite service's performance just as we discussed before.

Then we aggregate two aspects distribution context information (the aggregation is denoted as DisCtx for distribution context): structure-related importance and community-related replaceability of a service j, as follows:

$$\mathrm{DisCtx}_j = (1 - \rho) \times \mathrm{IMP}_{j,\mathrm{cs}} + \rho \times \mathrm{Rep}_{j,C} \tag{22}$$

Here, ρ is a weight attached to the structure-related importance and community-related replaceability of service j to indicate the significances. They are determined according to user's preferences.

If $\rho = 0$ indicates only the structure-related importance is employed, otherwise if $\rho = 1$ indicates only the community-related replaceability is employed, reversely. If $0 < \rho < 1$ indicates that we combine structure-related importance and community-related replaceability to distribute the score from a composite service to its component services.

5.3 Distributing score

After invoking the composite service, the QoS deviation of each component service(denoted as $Q^{\Delta}_{j,g}$) can be calculated during this invocation. Based on $Q^{\Delta}_{j,g}$ and distribution context information, which is discussed above, of the component service, its score can be obtained. In addition, some of these component services are still composite services, so this kind of distribution continues until all component services are atomic.

The basic idea is that each component service obtains its score according to its contribution to the quality deviation of the composite service. Suppose $s_1, \ldots, s_{|\mathrm{cs}|}$ are component services of the composite service cs which is invoked by a user k. From the gth QoS attribute point of view, we can calculate the deviation of gth QoS parameter as follows:

$$Q^{\Delta}_{j,g} = Q^{av}_{j,g} - Q_{j,g} \tag{23}$$

Here, $Q^{\Delta}_{j,g}$ is the value of the difference between and $Q_{j,g}$ and $Q^{av}_{j,g}$. Assume that the gth QoS parameter is negative (For the sake of simplicity, we only consider the negative QoS parameter in this paper, and our model can be easily to extended to other type's QoS parameters), If the gth QoS parameter is better than its expected average value, then $Q^{\Delta}_{j,g} > 0$, otherwise, $Q^{\Delta}_{j,g} < 0$. After calculating each deviation of QoS parameters, we can easily aggregate service j's performance deviation as:

$$Q^{\Delta}_{j} = \sum_{g=1}^{|Q_j|} w_g \times (Q^{\Delta}_{j,g}) \tag{24}$$

Similarly, we can also calculate the performance deviation of composite service cs as follows:

$$Q^{\Delta}_{\mathrm{cs}} = \sum_{g=1}^{|Q_j|} w_g \times (Q^{\Delta}_{\mathrm{cs},g}) \tag{25}$$

Here, $Q^{\Delta}_{\mathrm{cs},g}$ is the value of the difference between $Q_{\mathrm{cs},g}$ and $Q^{av}_{\mathrm{cs},g}$. The composite service's score is distributed according to component services' contributions to composite service performance (either increase or decrease). Then we can obtain the score of each component service thorough following three steps:

Step1: Calculate each component service's deviating score according to the distribution context (DisCtx) of every component service and the ratio of its performance deviation with composite service's performance deviation, called deviation score, denoted as $DeScore$:

$$\mathrm{DeScore}^{v}_{\mathrm{cs},j} = \mathrm{Score}^{v}_{k,\mathrm{cs}} \times \frac{Q^{\Delta}_{j}}{|Q^{\Delta}_{\mathrm{cs}}|} \times \frac{\mathrm{DisCtx}_j}{\sum_{j=1}^{|\mathrm{cs}|} \mathrm{DisCtx}_j} \quad (26)$$

Step2: Calculate the reminder score of composite service by adding each component service's minus deviation score and composite service's score, called reminder score, denoted as $RScore$:

$$\mathrm{RScore}^{v}_{k,\mathrm{cs}} = \mathrm{Score}^{v}_{k,\mathrm{cs}} + \sum_{j=1}^{|\mathrm{cs}|} (-\mathrm{DeScore}^{v}_{\mathrm{cs},j}) \quad (27)$$

Step3: Obtain each component service's score after each invocation, as follows:

$$\mathrm{Score}^{v}_{\mathrm{cs},j} = \mathrm{DeScore}^{v}_{\mathrm{cs},j} + \frac{\mathrm{DisCtx}_j}{\sum_{j=1}^{|\mathrm{cs}|} \mathrm{DisCtx}_j} \times \mathrm{RScore}^{v}_{k,\mathrm{cs}} \quad (28)$$

Here, $\mathrm{Score}^{v}_{\mathrm{cs},j}$ is indicate the score of composite service cs distributing to its component service j in vth invocation. The above procedure can distribute the score to every component service in a fair manner, which is guaranteed by the following claim.

Claim 2 *The quality deviation and context based approach can realized a fair score distribution.*

Proof We first consider a basic case where all component services have only one QoS parameter. As the services perform better than user expected, it will obtain a positive $\mathrm{DeScore}^{v}_{\mathrm{cs},j}$. This indicates that a component service is awarded for its good performance. As the services perform worse than user expected, it will obtain a negative $\mathrm{DeScore}^{v}_{\mathrm{cs},j}$. This indicates that a component service is penalized for its poor performance. As the services perform as user expected, it will obtain a a zero $\mathrm{DeScore}^{v}_{\mathrm{cs},j}$. This indicates that a component service neither awarded nor penalized. □

In addition, it is clear from formula (26) that the better/worse performance a service has, the higher/lower $\mathrm{DeScore}^{v}_{\mathrm{cs},j}$ it obtains. It is also clear from formula (26) and (28) that the service should be awarded more if it has higher context value while it performance as better as other services; the service should be penalized more if it has higher context value while its performance is as worse as other services. Similarly, the service should be awarded less if it has lower context value while its performance is as better as other services; the service should be penalized less if it has lower context value while its performance is as worse as other services. The similar analysis can be conducted when component service have multiple QoS parameters. We omit the detailed proof here due to space limitation.

6 Experiments

In this section, we evaluate the effectiveness of our distributing approach by conducting extensive experiments. All experiments are performed on a PC with 2.2 GHz Intel Pentium Duo2 CPU, 2048M of RAM, Microsoft Windows XP Operating System, J2SKD 1.6. In the section of related works, we have introduced that there are two works which focus on composite service's feedback (utility score) distribution. However, the work [12], which assigns each component service an importance value by user, is too subject and therefore it's hard for us to compare the approach proposed in [12] with our approaches (hard to assign importance to component services to ensure fair comparison). Thus, we compare our approach with the work 2PP method [14], which distributes the utility score of component services based only on their performance. The first one evaluates the structure-related impact of a composite service. Namely, each component service which has a variance location in a composite service may perform different contributions to the performance of the whole composite service's performance. The second one evaluates the community-related replaceability impact of a composite service. Each component service has different replaceability in its community, and the component service' replaceability value decides whether it can be easily replaced or not. Lastly, we use a hybrid method, which combines the structured-related importance and community-related replaceability, to distribute the component service's utility score to its component services to verify our he fairness of our method and a series of performance evaluation applies on this hybrid method is also given accordingly.

6.1 Impact of structure

Before considering the impact of structure-related importance on composite service's score distribution, we vary influence to the structure-related importance of component services. We use application scenario as an example, and assign each component service an initial reputation value by

random manner. Each component service is given an initial reputation value (0.2, 0.8, 0.2, 0.4, 0.7, 0.4, 0.9, and 0.6). The effects of the on the component service' structure-related importance is illustrated in Table 3. When $\alpha = 0$ and $\alpha = 1$, we can see in the Table 3 that the hybrid importance [$d^{R\alpha}$: c.f. Eq. (16)]: combines centrality score of degree and strength) value is equivalent to the degree only and strength only value, respectively. By further observation, when $0 < \alpha < 1$, the hybrid importance value is more than degree only value and lower than strength only value. When $\alpha > 1$,the hybrid importance value is more than strength only value. For simplicity and objectivity, in our next series of experiments, we set $\alpha = 0.5$ to split the difference of degree only value and strength only value.

Then, we consider that the composite service (cs) has fifteen component services, and all of them have the same advertised QoS performance. We generate the composite service by a random matrix and check whether it satisfies our requirements. In the next two evaluation, we also use the same method to generate composite services. We divide the component services into three groups, namely, importance high (IMP_H), importance middle (IMP_M), and importance low (IMP_L), according to their structure importance value. The IMP_H, IMP_M, and IMP_L indicate the hybrid structure-related importance of component services larger 0.8, between 0.4 and 0.8, and smaller than 0.4, respectively. In our experiment, we degrade or improve the performance of those three types' services 20, 50, and 80 % relative to their advertised QoS performance value. Note that, considering that web service run in a dynamic environment, 20, 50, and 80 % are approximate settings which indicate the fluctuations are within 20, 50, and 80 %, respectively. Meanwhile, we also set every service has a normal performance fluctuation within 5 %. Then we can plot its influences to the whole composite service's performance. The X-axis and Y-axis represent the number of executions and average percentages of component services degrading or improving the whole composite service's performance (IR: improvement ratio and DR: degradation ratio).

Figure 7a, b show that the average performance of component services improves and degrades 20 % relative to their advertised performance. We can observe that the group of component service which it's IMP_H has higher influences than the group of IMP_L. The remaining figures (Fig. 7c–f), where average performance improves and degrades 50 and 80 %, respectively, show the same meanings as Fig. 7a, b. From the Fig. 7, we can obviously see that the component services which degrades or improves the higher value relative to their advertised values may cause higher influences on the composite service cs. In addition, it is also shown that the component services with larger runtime performance fluctuation have the larger influence on the composite service quality by comparing Fig. 7a, b, with Fig. 7c, d or e, f. Therefore, we can safely conclude that the structure-related context should be considered in process of score distribution from a composite service to its component services. This because of services with different locations in a composite service have different contributions to the composite service's performance.

6.2 Impact of replaceability

In this part of the experiment, we study the replaceability impact on the whole composite service's execution. Assume that the composite service cs consists of ten component services, each of which belongs to their respective communities, i.e., cs has ten communities. The replaceability score of component services from the first to the tenth is different values varying from 0.1 to 1, as 0.1 as one step, respectively. Three different communities' size are considered, which contain 100, 1,000, and 10,000 member services, respectively. We consider the failure probability and success replaceability ratio of services with different replaceability scores in different communities. Here, the failure probability indicates that the component services' performance fluctuation violates the composite service's QoS constraints. And the success replaceability ratio indicates that the component service can success replace by other services which are in the same community. We also assume that the services with high replaceability score may have the larger fluctuation of performance because of they can be dominated by more services with the community (i.e., the rank of expected performance is low than others). For simplicity, we set the performance fluctuations of services is proportional to their replaceability score. We execute 100 times as a round and calculate statistic probability or ratio to plot in Fig. 8. Figure 8a shows that the failure probability increases, while the component services replaceability score changing from 0.1 to 1.0. The community size meanwhile is also influencing the failure probability, where the community with a larger size can reduce the failure probability but the range of reducing is limited. Figure 8b shows that the services with lower replaceability score can be easily replaced, corresponding with Fig. 8a. We can conclude that the component service with higher dominating score is more important than the one with lower dominating score in a composite service. This is because if a service with lower replaceability fails, it is hard to replace it without violating the whole composite service's QoS constraints, and the higher replaceability services will be on the contrary.

6.3 Hybrid distribution

We now apply our distribution framework to complex composite service cs. The component services can be divided into five groups (i.e., Consistently High, Consistently Low,

Table 3 Impact of tuning parameter on structure-related importance

Service	d	d^R	d^{Ra} when $\alpha =$			
			0	0.5	1	1.5
s_1	0.43	0.45	0.43	0.44	0.45	0.46
s_2	0.29	0.36	0.29	0.32	0.36	0.40
s_3	0.29	0.29	0.29	0.29	0.29	0.29
s_4	0.29	0.31	0.29	0.30	0.31	0.32
s_5	0.57	0.69	0.57	0.63	0.69	0.76
s_6	0.22	0.35	0.22	0.29	0.35	0.45
s_7	0.05	0.10	0.05	0.08	0.10	0.15
s_8	0.14	0.18	0.14	0.16	0.18	0.21

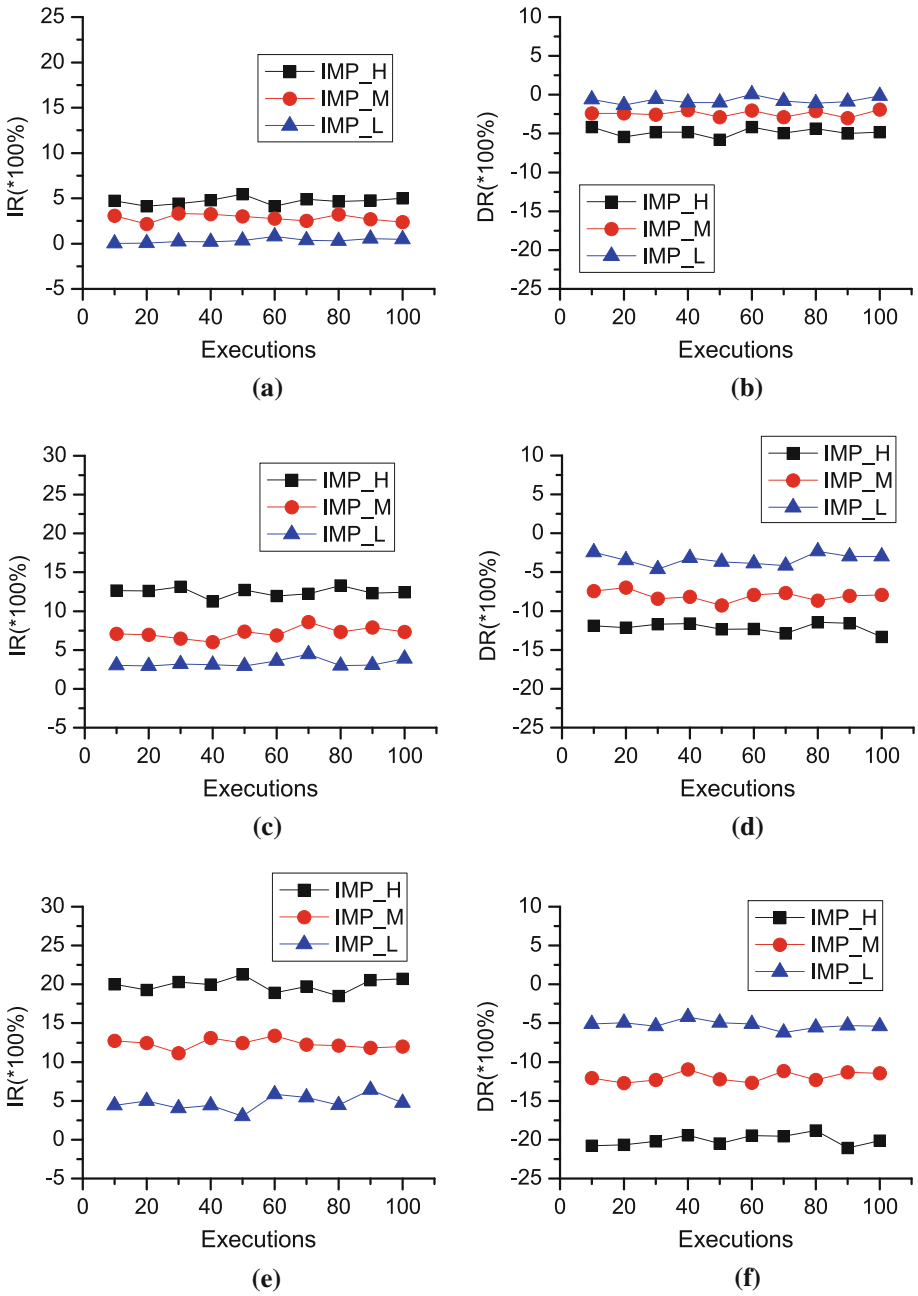


Fig. 7 Impact of importance to whole performance of cs. **a** Improvement 20 %, **b** degradation 20 %, **c** improvement 50 %, **d** degradation 50 %, **e** improvement 80 %, **f** degradation 80 %

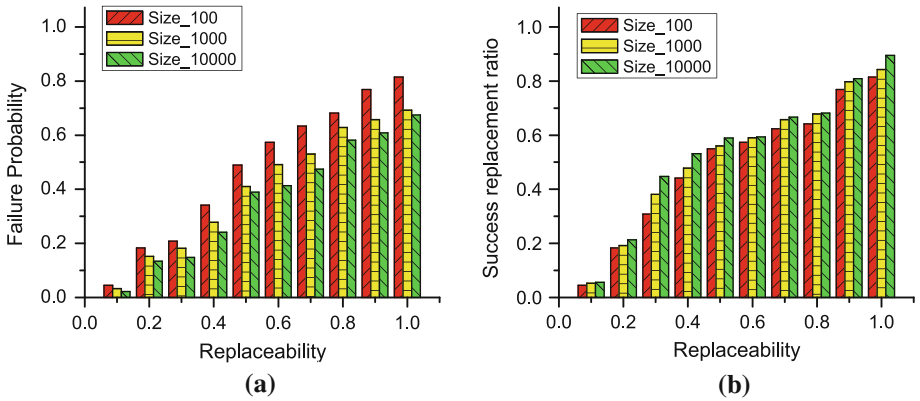


Fig. 8 Impact of community-related replaceability. **a** Failure probability, **b** success replacement ratio

Degrades from high to low, upgrades from low to high, and fluctuation) according to their performance. And each group has at least two component services, of which the importance is high and low, respectively. Both of those two services within the same group change their performance simultaneously and in the same extent. We use our approach to calculate the score on each group to obtain two types of scores (IMP_H and IMP_L), respectively. At the same time, the 2PP method can be used to calculate the score on those two services. However, it will obtain the same score, because it doesn't consider the distribution context of component service when distributing score from a composite service to its component services. Component services with consistently high performance always meet their advertised QoS promises. However, component services with consistently low performance always violate their advertised QoS promises. Component services with performance upgrading from low to high (high to low) always change their performance from bad to good (good to bad). Component services with fluctuating performance have oscillatory performance over a period of time.

Figure 9 shows the scores of component services when different approaches are applied. The X-axis represents the number of invocations of the composite service cs and the Y-axis indicates the score (Fig. 9a indicates the score of composite service cs) of the component service. Figure 9b shows that the component service with high distribution context value obtains high score than the component service with low distribution context value even though they have identical performance, since both of these two services have consistently high behaviors, and the service with high distribution context value contributes more than the service with low distribution context value. Meanwhile, in Fig. 9c, the component service with low distribution context value obtains a relative high score, since both of these two services with consistently low behaviors, have smaller influences than the service with high distribution context value, even though they have the same runtime performance. The remaining figures in Fig. 9 also illustrate the same meaning. We clearly conclude that the component service with high distribution context value should high responsibilities for composite service's invocation. If the component service has high distribution context value, it will get (lose) more due to its better performance (worse performance). However, the scores of component services obtained by the 2PP_N method in each groups are between IMP_H and IMP_L. It is obvious that this kind of score can't fairly reflect contribution of a component service to the performance of the composite service when both component services have the same performance.

Based on the observed results above, it is safe to conclude that our approach is better than 2PP_N in terms of the way of distributing the score to component services. Since we have considered the context of component services, our approach is obviously fairer and more accurate compared to 2PP_N method.

6.4 Performance evaluation

In the last part of experiments, we investigate the performance of our distribution approach variance the number of component services of the composite service and the size of the community.

First, we vary the number of component services of the composite service from 10 to 100, and the size of community from 100 to 10,000 (ref. Fig. 10). Then we combine two parts to assess the performance of our approach by setting one dimension as constant and one dimension as variable. Runtime overheads mainly involve the cost of calculating the centrality score, retrieving the dominating score and time it takes to distribute the composite service utility score to all of component services by the hybrid approach. The parameter is set as the same in hybrid experiments above. We design two cases to assess our runtime cost. We vary the number of component services from 10 to 100, and investi-

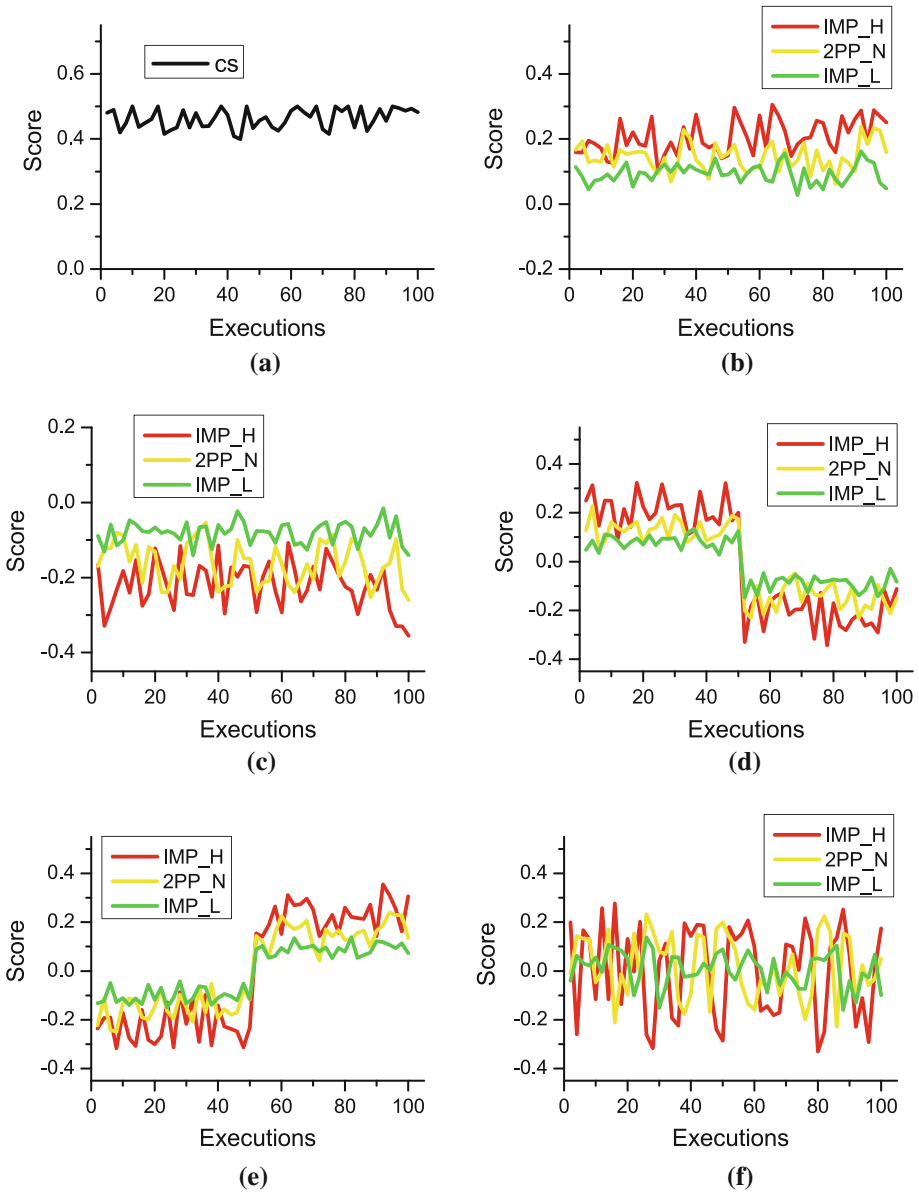


Fig. 9 Distributing the score based on hybrid importance. **a** Composite service, **b** consistently high, **c** consistently low, **d** degrade from high to low, **e** upgrade from low to high, **f** fluctuations

gate the community's size are 100, 5000 and 10,000, respectively in Fig. 10a and vary the size of community from 1,000 to 10,000, and investigate the number of component services are 10, 50,100 in Fig. 10b, respectively. Figure 10 shows that the overhead increases with the number of component services and the size of community. Meanwhile, as shown in Fig. 10a, even though the community size is 10,000 and the number of component service is 100, the approach can distribute the score within 150 ms. Secondly, we set the community size from 100 to 10,000 and investigate the runtime cost when the number of component services is 10, 50 and 100, respectively. It has the same cost as the first case's. Therefore, we can conclude that our distribution approach is indeed efficient.

7 Conclusions

In this paper, we have proposed an importance-based framework to fairly distribute the score of a composite service to its component services. Two facets of context have been introduced to ensure the fairness of the distribution. One is structure-related importance which is computed by graph theory; the other is community-related replaceability which is computed via dominating relationships of a community. In summary, our approach achieves a fair score distribution, which is significant for reputation computation in the context of service composition.

In the future, we will investigate the effect of corporation of multiple component service to fulfill a requirement of

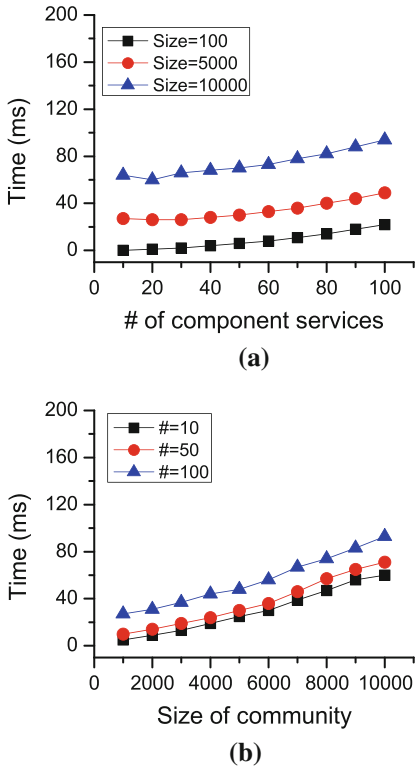


Fig. 10 Performance evaluation. **a** Performance of structure-related, **b** performance of community-related

the composite service. In addition, more context information which can influence the fairness of the distribution will also be considered continuously.

Acknowledgments This work has been supported by the Natural Science Foundations of China with the projects 60873234, 61003044, the Natural Science Foundation of Jiangsu Province under Grant No. BK2010257, and State Key Laboratory of Software Engineering (SKLSE).

References

1. Papazoglou MP, Georgakopoulos D (2003) Service-oriented computing. Commun ACM 46(10):25–65
2. Haesen R, Snoeck M, Lemahieu W, Poelmans S (2008) On the definition of service granularity and its architectural impact. In: Proceedings of international conference on advanced information systems engineering (CAiSE'08), pp 375–389
3. Web Services Business Process Execution Language Version 2.0, OASIS Standard, 11, April, (2007). http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html. Accessed 5 March 2011
4. Zaki M, Bouguettaya A (2009) RATEWeb: reputation assessment for trust establishment among web services. VLDB J 18:885–911
5. Li Z, Wang S (2005) The foundation of E-commerce: social reputation system-a comparison between American and China. In: Proceedings of international conference on electronic commerce (CEC'05), ACM Press, New York, pp 230–232
6. Chiu DKW, Leung H-F, Lam K-M (2009) On the making of service recommendations: an action theory based on utility, reputation, and risk attitude. Exp Syst Appl 36:3293–3301
7. Li X, Liu L (2004) PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities. IEEE Trans Knowl Eng 16(7):843–857
8. Maximilien EM, Singh MP (2002) Conceptual model of web services reputation. SIGMOD Rec 31(4):36–41
9. Binder BW, Drago ML, Ghezzi C (2009) REMAN:a pro-active reputation management infrastructure for composite web service. In: IEEE international conference on software engineering (ICSE'2009), May 16–24, pp 623–626
10. Binder BW, Drago ML, Ghezzi C (2008) Transparent reputation management for composite web service. In: IEEE international conference on web services (ICWS'2008), September 23–26, pp 621–628
11. Conner W, Lyengar A, Mikalsen T (2009) A trust management framework for service-oriented environments. In: International conference on world wide-web (WWW'2009), April 20–24, pp 891–900
12. Nepal S, Malik Z, Bouguettaya A (2009) Reputation propagation in composite services. In: IEEE international conference on web services (ICWS'2009), September, pp 295–302
13. Li L, Wang Y (2008) A trust vector approach to service-oriented application. In: IEEE international conference on web services (ICWS'2008), September 23–26, pp 621–628
14. Liu A, Li Q, Huang L, Wen S, Tang C (2010) Reputation-driven recommendation of services with uncertain QoS. In: Proceedings of IEEE Asia-Pacific services computing conference (APSCC'2010), Hang Zhou, Dec., pp 6–10
15. Dellarocs C (2003) The digitalization of word-of-mouth: promise and challenges of online feedback mechanism. Manag Sci 49(10):1407–1424
16. Sing MP, Huhns MN (2005) Service-oriented computing. Wiley Online Library, New York
17. Sabater J, Sierra C (2003) Reputation and social network analysis in multi-agent systems. In: Proceedings of the first international joint conference on autonomous agents and multiagent systems, Bologna, Italy, pp 475–482
18. Huynh TD, Jennings NR, Shadbolt NR (2006) Certified reputation: how an agent can trust a stranger. In: Proceedings of the fifth international joint conference on automomous agents and multiagent systems, Japan, pp 1217–1224
19. Kamvar SD, Scholosser MT, Garcia-Molina H (2003) The EigenTrsut algorithm for reputation management in P2P networks. In: International conference on world wide web (WWW'03), May 20–24
20. Zhou R, Hwang K (2007) PowerTrust: a robust and scalable reputation system for trusted peer-to-peer computing. IEEE Trans Parallel Distrib Syst 18(4):472–560
21. Buchigger S, Boudec J-YL (2004) A robust reputation system for P2P and mobile ad-hoc networks. In: Proceeding of second workshop economics of P2P systems
22. Brinklov M, Sharp R (2007) Incremental trust in grid computing. In: Proceedings of the seventh IEEE international symposium on cluster computing and the grid, pp 135–144
23. Azzedin F, Maheswaran M, Mitra A (2006) Trust brokering and its use for resource matchmaking in public-resource grids. J Grid Comput 4(3):247–263
24. Kreps DM, Wilson R (1982) Reputation and imperfect information. J Econ Theory 27(2):253–279
25. Holmstrom B (1999) Managerial incentive problems: a dynamic perspective. Rev Econ Stud 66(1):169–182
26. Huberman BA, Wu F (2004) The dynamics of reputation. J Stat Mech Theory Exp **2004**:P04006
27. Zeng L, Benatallsh B, Ngu AHH, Dumas M, Kalagnanam J, Chang H (2004) QoS-aware middleware for web services composition. IEEE Trans Softw Eng 30(5):311–327

28. Berbner R, Spahn M, Repp N, Heckmann Q, Steinmetz R (2006) Heuristics for QoS-aware web service composition. In: IEEE international conference on web services (ICWS), pp 72–82
29. Stein S, Payne TR, Jennings NR (2009) Flexible provisioning of web service workflows. ACM Trans Internet Technol 9(1), Article 2, p 45
30. Cambridge Dictionary Online. http://dictionary.cambridge.org/
31. Wen S, Li Q, Yue L, Liu A, Tang C (2010) Towards fair reputation propagation from a composite service to its component services. In: IEEE international conference on E-business engineering (ICEBE'2010), Nov. 10–12
32. Liu A, Li Q, Huang L, Xiao M, Liu H (2008) QoS-aware scheduling of web service. In: Proceedings of international conference on web-based information management (WAIM'08), pp 171–178
33. Opsahl T, Agneessens F, Skvoretz J (2010) Node centrality in weighted networks: generalizing degree and shortest paths. Soc Netw 32:245–251
34. Alonso G, Casati F, Kuno H, Machiraju V (2004) Web services: concepts, architectures and applications. Springer, Berlin