Int. J. Web and Grid Services, Vol. x, No. x, 1–22

Run-Time Resolution of Service Property Conflicts in Web Service Composition

Jiuyun Xu, Xiao Ning

School of Computer & Communication Engineering,
China University of Petroleum, Qingdao, China
E-mail: jiuyun.xu@ieee.org; xiaon2013@gmail.com

Stephan Reiff-Marganiec

Department of Computer Science, University of Leicester, UK
E-mail: srm13@leicester.ac.uk

Qiang Duan

Information Sciences and Technology Department, Pennsylvania State University, USA
E-mail: qxd2@psu.edu

Zibin Zheng

School of Data and Computer Science, Sun Yat-sen University, China
E-mail: zibin.gil@gmail.com

Abstract: With rapid development of Web service technologies, service composition has become a common approach to realizing complex business processes. Due to the large number of services developed and deployed independently by various providers, undesirable interactions between properties of different service components may occur when they are composed into a composite service. Such service property conflicts become a serious obstacle for service composition to meet users' requirements. Although traditional feature interaction techniques may prevent some property conflicts in service design stage, many conflicts occur during execution based on certain run-time data; therefore must be resolved online. In this paper, we propose a scheme to address the problem of run-time resolution of service property conflicts. We first formulate the conflict resolution problem with a bi-objective optimization model based on user's revenue, which represents the QoS and success rate of a service. Then we solve the bi-objective optimization model to obtain a set of Pareto solutions and rank the solutions to identify the optimal one, which gives the best roll back strategy and alternative service plan for resolving a service property conflict. We also implement the proposed scheme in a prototype of online shopping application and evaluate performance of the scheme through experiments. The obtained experimental results indicate that our scheme is effective and efficient in resolving service property conflicts at runtime.

Keywords: Web service composition; service property conflict; bi-objective optimization; runtime.

Reference to this paper should be made as follows: Xu, J., Xiao N., Duan Q., Reiff-Marganiec S. and Zheng Z. (2015) 'Run-Time Resolution of Service Property Conflicts in Web Service Composition', *Int. J. Web and Grid Services*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Jiuyun Xu received the Ph.D. degree in Computer Science from China University of Posts and Telecommunications in 2004. He is now a Professor of China University of Petroleum (Eastern China). His research interests include service computing and feature interaction. He has published in excess of 30 papers.

Xiao Ning received her Master Degree in Computer Application at China University of Petroleum. She is an engineer of China National Offshore Oil corp.

Stephan Reiff-Marganiec received the Ph.D. degree in Computer Science from the University of Glasgow (UK) in 2002. He is now a Senior Lecturer at the University of Leicester where his research focuses on service computing, feature interaction and business processes. He has published in excess of 50 papers and co-edited a number of books and conference proceedings.

Qiang Duan is an Associate Professor of Information Sciences and Technology Department at Pennsylvania State University Abington College. His current research areas include the next generation Internet, network-as-a-service, software-defined networking, network virtualization, Cloud computing, and Web services. He has published over 70 technical papers in international journals and conference proceedings and authored 5 book chapters. Dr. Duan is the editor-in-chief of Journal of Advanced Computer Science and Technology, an associate editor for 6 international research journals, and a regular reviewer for some prestigious journals including IEEE JSAC, IEEE TNSM, IEEE TPDS, IEEE TCC, ACM TAAS, etc. He has also served on the technical program committees for numerous conferences including GLOBECOM, ICC, ICCCN, AINA, WCNC, etc. Dr. Duan received his Ph.D. degree in electrical engineering from the University of Mississippi. He holds a B.S. degree in electrical and computer engineering and a M.S. degree in telecommunications and electronic systems.

Zibin Zheng is an associate Professor at Sun Yat-Sen University, China. He received his Ph.D. degree from The Chinese University of Hong Kong in 2011. He received Outstanding Thesis Award of The Chinese University of Hong Kong at 2012, ACM SIGSOFT Distinguished Paper Award at ICSE'10, Best Student Paper Award at ICWS'10, and IBM Ph.D. Fellowship Award 2010-2011. His research interests include service computing, cloud computing, and software engineering.

1 Introduction

The past few decades have witnessed the proliferation of Web services developed and provisioned by various service providers. In a competitive Web service market, multiple services developed and deployed by different vendors often provide the same service functionality. Service providers tend to add extra features to their services in order to gain competitive advantage. In addition, services with the same functionality may be hosted on heterogeneous platforms by different providers; thus showing different run-time features when being executed. Such features are referred to as properties of Web services.

Service composition has been widely applied to form business processes for meeting diverse user requirements. When individual Web services are composed, their properties may interact with each other in unexpected and often undesirable ways, which will degrade Quality of Service (QoS) or even cause execution failure of the composite service. We refer to such undesirable interactions between service properties as Web service property conflicts. With rapid increasing in the number of available Web services that may be involved in service composition, service property conflicts become more likely to occur and their negative effect on service performance becomes more severe. Therefore, how to resolve service property conflicts in Web service composition is an important research problem.

Currently available approaches to addressing the Web service property conflict issue are mainly either offline methods using formal verification Michael et al. (2004, 2007) or online methods based on behavior descriptions Zhang et al. (2006, 2007b,a). However, these methods are constrained by some limitations. First, independence of Web service development and distribution of service deployment make internal logics of services are often unavailable; therefore it is difficult to obtain accurate service descriptions required by the formal verification methods for resolving service property conflicts. Second, many service properties are determined by data that become available only during execution, for example data about user profiles and service host environments. This makes some property conflicts only appear at run-time thus cannot be detected and resolved on the service design stage with offline methods. Therefore, it is important to development technologies for run-time detection, recovery, and resolution of Web service property conflicts Babak et al. (2004).

As an example of run-time service property conflicts, we consider an online shopping system that consists of atomic services of online shops, bank payment, and product delivery. If a customer is a VIP member of the provider A of delivery service (thus receiving discount for delivery cost), then the service composition at the design stage will choose provider A for the delivery service to minimize the total service cost. Suppose the online shop service has a property of free delivery provided by B for any customer whose purchase amount is greater than a threshold, then such a property might cause a conflict with the delivery service provided by A when the customer purchases more than a certain amount. Occurrence of such a conflict depends on the value of some run-time data (in this example customer purchase amount). The conflict will degrade service performance (cause extra service cost in this case) unless it is detected and resolved at run-time.

In this paper we attempt to tackle the problem of run-time property conflicts of Web services. A solution to this problem comprises two aspects: run-time conflict detection of service properties and run-time resolution of service property conflicts. In our previous work we developed a run-time conflict detection technique based on the situation calculus Xu et al. (2010) and proposed the STRIPS method for detecting service property conflicts Xu et al. (2011). Therefore, in this paper we focus our study on run-time resolution of

service property conflicts and propose a novel online approach to conflict resolution based on bi-objective optimization. Specifically we make the following contributions in this paper.

- We formulate run-time resolution of Web service property conflicts as a bi-object optimization problem.
- We propose a bi-objective model to obtain a set of Pareto solutions to the problem and develop an algorithm to identify the optimal roll-back strategy and alternative service plan for resolving service conflicts.
- We implement the proposed resolution scheme in a prototype and conduct experiments on the prototype to verify the effectiveness of our proposed approach.

The remainder of the paper is organized as follows. We first describe a system architecture for run-time service management and present an illustrative example scenario in Section 3. In Section 4 we formulate service property conflict resolution as a bi-objective optimization problem and propose algorithms for solving this problem to obtain optimal resolution strategy. Experimental results are reported in Section 5 to verify effectiveness of the proposed model and algorithms. We discuss related work in Section 6 and draw conclusions in Section 7.

2 An Architectural Framework of Run-Time Resolution for Service Property Conflicts

A business process may be constructed through Web service composition with two steps. First a set of abstract services (or called tasks) are selected and composed to form a workflow for meeting the user's requirement. We assume that there are multiple specific services (referred as services for simplicity in the rest of this paper) available for realizing each task in the workflow; that is, there is a set of available services that all provide the functionalities required by the task. Then the second step is to select one service for each task of the workflow to construct a service plan; thus forming a composite service for realizing the workflow. There may be multiple service plans for a given workflow based on different service composition strategies.

We first explain the following concepts then describe the system architecture of service management with run-time detection and resolution of property conflicts.

- *Service plan* p is a sequence of atomic services that implement all the tasks of a workflow to meet user's business requirement.
- *Plan utility* u_p is the utility value of a service plan, which can be computed using a suitable method during the process of constructing the service plan.
- *Service model* M is a collection of all possible service plans of a workflow for achieving a user's requirements. It is denoted as a set $M = \{p_1, p_2, \cdots, p_k\}$. The plan $p \in M$ with the maximal utility value will be the preferred plan for executing the composite service; all others plans are candidate plans.
- *Executed plan* p_e is the part of a service plan that has been executed. it can be represented as a sequence of pairs $\{\langle s_1, ws_1\rangle, \langle s_2, ws_2\rangle, \ldots, \langle s_n, ws_n\rangle\}$. Each pair $< s_i, ws_i >$ denotes the initial state s_i before the execution of service ws_i. In the sequence, s_1 is the

initial state before any service is executed, s_n is the initial state before the last service is executed. Each $\langle s_i, ws_i \rangle$ for $1 < i < n$ is the only successor of $\langle s_{(i-1)}, ws_{(i-1)} \rangle$ and the only predecessor of $\langle s_{(i+1)}, ws_{(i+1)} \rangle$. Furthermore, ws_i for $1 \leq i \leq n$ will be executed when s_i satisfies its preconditions.

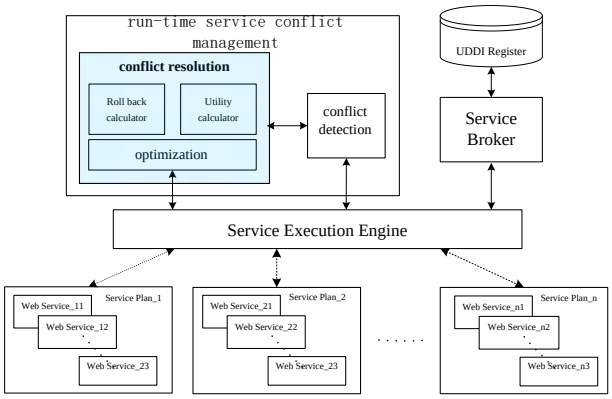


Figure 1: System architecture of service management with run-time conflict detection and resolution.

An architectural framework of run-time service management is shown in Figure 3. The key components of the system include a the *service broker*, a *service execution engine*, and a *conflict manager* that consists of a *conflict detector* and a *conflict resolver*.

The service broker maintains an UDDI registry where service providers register their services and publish service descriptions. A service description contains meta data that describes the capability and QoS of a Web service. Upon receiving a service request from a user with a requirement specification, the service broker starts a service composition process. The broker will obtain a service model that includes all possible service plans that meet the user's requirement and then evaluate the utility value for each plan. The service plan with the maximal utility value will be selected for execution. The execution engine orchestrates execution of the atomic services in the selected service plan. During service execution, the conflict detector keeps checking property interactions among individual services and informs the conflict resolver with a conflict description if any conflict is detected. The conflict resolver is responsible for coming up with an optimal strategy for resolving the conflict in order to recover the execution from failure caused by the conflict. Specifically, the process of conflict resolution performed at the conflict resolver has the following steps:

1. Calculate the roll-back service set and respective cost.

2. The bi-objective algorithm determines Pareto solutions for the service plan

3. The optimality algorithm determines the rank of the strategies; the top ranked one is the best strategy and others are candidates.

4. The execution engine in the system continues to execute the service plan by invoking Web services in line with the conflict resolution strategy.

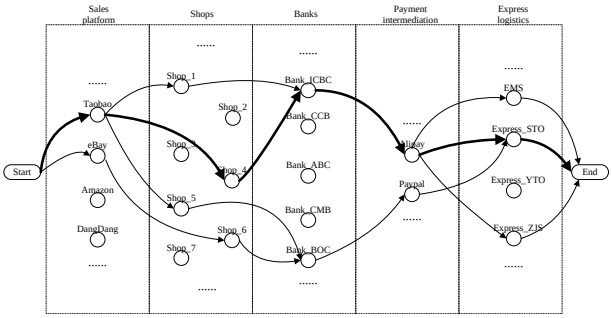


Figure 2: An illustrative scenario of run-time service property conflict.

As an illustrative scenario we consider an online shopping application in which the business process comprise the sequence of tasks – accessing sale platform, purchasing in shops, paying through a bank account and payment partner, and delivery via express logistic. Multiple services are available for each task, as shown in Figure 4. Descriptions of these available services are published at an UDDI registry. When a customer wants to start online shopping for purchasing some products, she submits a service request with some specification information to the service broker. Upon receiving a request, the broker searches available services to find all possible service plans and constructs a service model (the collection of all feasible service plans). The broker also evaluates the utility value of each service plan and choose the one with the maximal utility value as the preferred plan for execution. For example, six plans are feasible in this case, as shown in Figure 4, and their utility values are given in Table 2. Then the plan P_2 is selected by the broker as the result of service composition. The broker provides the service model and selected service plan to the service execution engine, and the latter will start invoking services by following the selected plan.

The execution engine collects data from invoked services during service execution. The collected data will be processed by the conflict detector to search run-time property conflicts that may degrade service performance. For example, the service customer is an VIP member of STO who receives discount for online shopping delivery; therefore STO service is selected in the preferred plan. However, Shop4 offers free EMS delivery when a customer's shopping amount is greater than a certain threshold, which is met by the current customer. Then there exists a conflict between the properties of Shop4 and STO services in the selected service plan P_2. Such a conflict has to be detected run-time because it is trigger by certain run-time data at a service (in this example the purchase amount at Shop4). The conflict resolver, when informed by the detector about a found conflict, will start a resolution process to come up with a roll back strategy and an alternative service plan. For example in this call the strategy is to roll back to Alipay service and then take P_3 as the alternative plan.

3 An Architectural Framework of Run-Time Resolution for Service Property Conflicts

A business process may be constructed through Web service composition with two steps. First a set of abstract services (or called tasks) are selected and composed to form a workflow

Table 1 Utility Values and Textual Representation of Plans

Plan (p)	Utility Value ($\mathcal{U}(p)$)	Services
P_1	2.5	Taobao, Shop1, ICBC, Alipay, STO
P_2	4.8	Taobao, Shop4, ICBC, Alipay, STO
P_3	3.6	Taobao, Shop4, ICBC, Alipay, EMS
P_4	3.0	Taobao, Shop4, BOC, Alipay, ZJB
P_5	4.3	Taobao, Shop5, BOC, Paypal, STO
P_6	3.5	eBay, Shop6, BOC, Paypal, STO

for meeting the user's requirement. We assume that there are multiple specific services (referred as services for simplicity in the rest of this paper) available for realizing each task in the workflow; that is, there is a set of available services that all provide the functionalities required by the task. Then the second step is to select one service for each task of the workflow to construct a service plan; thus forming a composite service for realizing the workflow. There may be multiple service plans for a given workflow based on different service composition strategies.

We first explain the following concepts then describe the system architecture of service management with run-time detection and resolution of property conflicts.

- *Service plan* p is a sequence of atomic services that implement all the tasks of a workflow to meet user's business requirement.
- *Plan utility* u_p is the utility value of a service plan, which can be computed using a suitable method during the process of constructing the service plan.
- *Service model* M is a collection of all possible service plans of a workflow for achieving a user's requirements. It is denoted as a set $M = \{p_1, p_2, \cdots, p_k\}$. The plan $p \in M$ with the maximal utility value will be the preferred plan for executing the composite service; all others plans are candidate plans.
- *Executed plan* p_e is the part of a service plan that has been executed. it can be represented as a sequence of pairs $\{\langle s_1, ws_1\rangle, \langle s_2, ws_2\rangle, \ldots, \langle s_n, ws_n\rangle\}$. Each pair $< s_i, ws_i >$ denotes the initial state s_i before the execution of service ws_i. In the sequence, s_1 is the initial state before any service is executed, s_n is the initial state before the last service is executed. Each $\langle s_i, ws_i\rangle$ for $1 < i < n$ is the only successor of $\langle s_{(i-1)}, ws_{(i-1)}\rangle$ and the only predecessor of $\langle s_{(i+1)}, ws_{(i+1)}\rangle$. Furthermore, ws_i for $1 \leq i \leq n$ will be executed when s_i satisfies its preconditions.

An architectural framework of run-time service management is shown in Figure 3. The key components of the system include a the *service broker*, a *service execution engine*, and a *conflict manager* that consists of a *conflict detector* and a *conflict resolver*.

The service broker maintains an UDDI registry where service providers register their services and publish service descriptions. A service description contains meta data that describes the capability and QoS of a Web service. Upon receiving a service request from a user with a requirement specification, the service broker starts a service composition process. The broker will obtain a service model that includes all possible service plans that meet the user's requirement and then evaluate the utility value for each plan. The service plan with the maximal utility value will be selected for execution. The execution engine orchestrates execution of the atomic services in the selected service plan. During service execution,

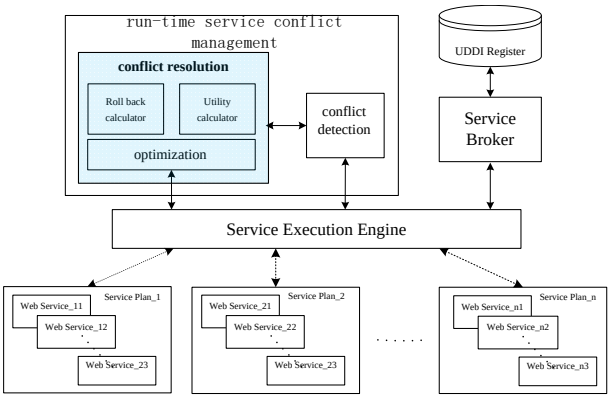


Figure 3: System architecture of service management with run-time conflict detection and resolution.

the conflict detector keeps checking property interactions among individual services and informs the conflict resolver with a conflict description if any conflict is detected. The conflict resolver is responsible for coming up with an optimal strategy for resolving the conflict in order to recover the execution from failure caused by the conflict. Specifically, the process of conflict resolution performed at the conflict resolver has the following steps:

1. Calculate the roll-back service set and respective cost.
2. The bi-objective algorithm determines Pareto solutions for the service plan
3. The optimality algorithm determines the rank of the strategies; the top ranked one is the best strategy and others are candidates.
4. The execution engine in the system continues to execute the service plan by invoking Web services in line with the conflict resolution strategy.

As an illustrative scenario we consider an online shopping application in which the business process comprise the sequence of tasks – accessing sale platform, purchasing in shops, paying through a bank account and payment partner, and delivery via express logistic. Multiple services are available for each task, as shown in Figure 4. Descriptions of these available services are published at an UDDI registry. When a customer wants to start online shopping for purchasing some products, she submits a service request with some specification information to the service broker. Upon receiving a request, the broker searches available services to find all possible service plans and constructs a service model (the collection of all feasible service plans). The broker also evaluates the utility value of each service plan and choose the one with the maximal utility value as the preferred plan for execution. For example, six plans are feasible in this case, as shown in Figure 4, and their utility values are given in Table 2. Then the plan P_2 is selected by the broker as the result of service composition. The broker provides the service model and selected service plan to the service execution engine, and the latter will start invoking services by following the selected plan.

The execution engine collects data from invoked services during service execution. The collected data will be processed by the conflict detector to search run-time property conflicts that may degrade service performance. For example, the service customer is an

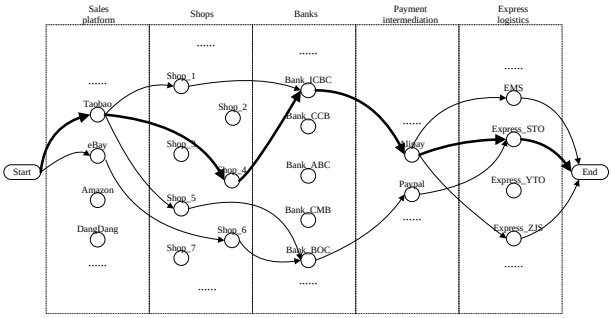


Figure 4: An illustrative scenario of run-time service property conflict.

Table 2 Utility Values and Textual Representation of Plans

Plan (p)	Utility Value ($\mathcal{U}(p)$)	Services
P_1	2.5	Taobao, Shop1, ICBC, Alipay, STO
P_2	4.8	Taobao, Shop4, ICBC, Alipay, STO
P_3	3.6	Taobao, Shop4, ICBC, Alipay, EMS
P_4	3.0	Taobao, Shop4, BOC, Alipay, ZJB
P_5	4.3	Taobao, Shop5, BOC, Paypal, STO
P_6	3.5	eBay, Shop6, BOC, Paypal, STO

VIP member of STO who receives discount for online shopping delivery; therefore STO service is selected in the preferred plan. However, Shop4 offers free EMS delivery when a customer's shopping amount is greater than a certain threshold, which is met by the current customer. Then there exists a conflict between the properties of Shop4 and STO services in the selected service plan P_2. Such a conflict has to be detected run-time because it is trigger by certain run-time data at a service (in this example the purchase amount at Shop4). The conflict resolver, when informed by the detector about a found conflict, will start a resolution process to come up with a roll back strategy and an alternative service plan. For example in this call the strategy is to roll back to Alipay service and then take P_3 as the alternative plan.

4 Run-Time Resolution for Service Property Conflicts

In this section we propose a run-time scheme for resolving service property conflicts in Web service composition. We assume that service property conflicts may be detected by some suitable method for example Xu et al. (2010); therefore we focus on determining a conflict resolution strategy based on the execution model M for meeting the user's requirements. The resolution scheme consists of two elements: a roll-back strategy that identifies part of the current execution plane that needs to undertake compensation operation; and a strategy for selecting an alternative execution plane that can complete the service for meeting user requirements.

We transform the service property conflict problem to a bi-objective optimization problem and then develop a bi-objective optimization model for obtaining a set of Pareto solutions. Then we rank the Pareto solutions to identify the optimal one, which includes the

Table 3 Description of web service QoS attributes

Quality Criteria	Description	Notation
Reliability(Rb)	The trait of the service being dependable or reliable	$q_{Rb}(ws)$
Response time(RT)	The expected delay in seconds between the moment when a request is sent and the moment when the result are received.	$q_{RT}(ws)$
Cost(C)	The fee that a service requester has to pay for invoking the service	$q_C(ws)$
Reputation(R)	A measure of its trustworthiness and defined as the average ranking given to the service by end users.	$q_R(ws)$
Safety(S)	Safety provided by the service when use it	$q_S(ws)$
Success Rate(SR)	The probability that a request is correctly responded.	$q_{SR}(ws)$

optimal roll-back strategy and alternative execution plan. The method consists of four basic steps: 1) identifying the expected utility of the service plan, 2) identifying the roll-back cost, 3) establishing the bi-objective model, and 4) solving the bi-objective problem.

4.1 User Revenue-Based Utility Model for Web Service Composition

In this subsection we develop a utility model based on user revenue for evaluating the quality of service plans as a basis of our resolution scheme for run-time service property conflicts. User revenue is the benefit that a user gains from a service in a specific deployment and running condition. User revenue may be influenced by multiple factors including service function, QoS, and usage mode. For a service with a certain function and specific usage mode, QoS is a critical impact factor of the user revenue. Therefore, we will first develop an QoS model from a user perspective, which is called U-QoS in this paper.

QoS is a broad concept that encompasses a number of non-functional properties such as availability, reliability, service cost, and reputation Justin et al. (2002). The U-QoS model can be presented as a vector $U - QoS = \langle q_{Rb}, q_{Rt}, q_C, q_R, q_S, q_{SR} \rangle$ with multiple parameters reflecting different aspects of service quality from a user perspective. The detailed descriptions of these parameters are given in Table 3. Among the three typical types of composite service plans with sequential, parallel, and conditional structures, we focus on sequential service plans in this paper. The formulas for calculating QoS attributes are shown in Table 4.

In order to model U-QoS for a service model that contains a list of service plans, we defined a matrix $Q = (q_{i,j}; 1 \leq i \leq n, 1 \leq j \leq m)$, where n is the number of composite service plans, m is the number of QoS attributes with the encoding *1=reliability, 2=response time, 3=cost, 4=reputation, 5=safety, 6=success rate*, and $q_{i,j}$ denotes the value of attribute *j* in plan *i*, $(1 \leq i \leq n, 1 \leq j \leq m)$. Therefore, in the matrix Q each row q_j corresponds to a service plan p and each column corresponds to an QoS attribute.

Table 4 Computation of Web services QoS Attributes in sequential execution

Quality Criteria	Aggregation Function
Reliability(Rb)	$\prod_{i=1}^{n} q_{Rb}(ws_i)$
Response time(RT)	$\sum_{i=1}^{n} q_{RT}(ws)$
Cost(C)	$\sum_{i=1}^{n} q_C(ws)$
Reputation(R)	$\sum_{i=1}^{n} q_R(ws)/n$
Safety(S)	$min\{q_S(ws_1), q_S(ws_2), \cdots, q_S(ws_n)\}$
Success Rate(SR)	$min\{q_{SR}(ws_1), q_{SR}(ws_2), \cdots, q_{SR}(ws_n)\}$

if $d_j = 1$,

$$q'_{i,j} = \begin{cases} q_{i,j} / \frac{1}{n} \sum_{i=1}^{n} q_{i,j}, & \text{if } \frac{1}{n} \sum_{i=1}^{n} q_{i,j} \neq 0 \text{ and } q_{i,j} / \frac{1}{n} \sum_{i=1}^{n} q_{i,j} < l_j; \\ l_j, & \text{else;} \end{cases} \quad (1)$$

if $d_j = 0$,

$$q'_{i,j} = \begin{cases} \frac{1}{n} \sum_{i=1}^{n} q_{i,j} / q_{i,j}, & \text{if } q_{i,j} \neq 0 \text{ and } \frac{1}{n} \sum_{i=1}^{n} q_{i,j} / q_{i,j} < l_j; \\ l_j, & \text{else;} \end{cases} \quad (2)$$

Some QoS attributes, such as service delay and cost, represent better quality with lower values; therefore are referred to as negative attributes in the paper. Other attributes are positive in the senses that higher values indicate better quality. In order to enable a uniform process for both type of attributes, we define a direction vector $D = (d_1, d_2, \cdots, d_i, \cdots, d_m)$, where $d_i = 1$ means the corresponding attribute is positive and $d_i = 0$ means the attribute is negative. The limitation vector is defined as $L = (l_1, l_2, \cdots, l_i, \cdots, l_m)$, where each l_i captures the maximum value of the attribute. For positive attributes, values are scaled according to (1). The values of negative attributes are scaled according to (2). In equations (1) and (2), $\frac{1}{n} \sum_{i=1}^{n} q_{i,j}$ is the average of a quality attribute in matrix Q. By applying the equations to each element of Q, we obtain matrix $Q' = (q'_{i,j}; 1 \leq i \leq n, 1 \leq j \leq m)$.

In order to reflect the difference in users' requirements regarding their preferences to QoS attributes, we define a weight vector $w = (w_1, w_2, \cdots, w_i, \cdots, w_m)$ where $\sum_1^m w_i = 1$, $0 \leq w_i \leq 1, i = 1, 2, \cdots, m$. Applying the weight vector to the scaled quality matrix Q' we can obtain a weighted QoS matrix as shown in (3).

$$QoS(ws_i) = \sum_{j=1}^{m} w_j * q'_{i,j}. \quad (3)$$

In addition to service QoS, the successful switch rate of a service plan is another important factor that impacts the user revenue from the service. Successful switch rate is the probability that the service can be completed for meeting the user requirement by successfully switching to an alternative service plan when an unexpected matter occurs during execution of the current plan. It is denoted as $RS(ws) = T_s/T_{sum}$, where T_s is the

sum of successful execution times and unsuccessful execution but successful change times, meanwhile, T_{sum} is the total execution times of the plan.

U-QoS model and service successful switch rate are two dimensions of user revenue and can be normalized to form the user revenue-based utility model for services. From (3) we obtain QoS matrix $QoS = (QoS_1, QoS_2, \cdots, QoS_m)^T$. Then we can get the normalized QoS matrix $QoS' = (QoS'_1, QoS'_2, \cdots, QoS'_m)^T$ by using equation (4)

$$QoS'_i = \begin{cases} \frac{QoS_i - QoS_{min}}{QoS_{max} - QoS_i}, & QoS_{max} - QoS_i \neq 0; \\ 1, & QoS_{max} - QoS_i = 0; \end{cases} \tag{4}$$

Similarly, successful switch rate $RS = (RS_1, RS_2, \cdots, RS_m)^T$ is normalised to get $RS' = (RS'_1, RS'_2, \cdots, RS'_m)^T$. Then the utility value of a composite plan can be obtained by equation (5). The procedure for calculating utility values of a set of service plans in a service model is given in Algorithm 1.

$$U = QoS' + RS' \tag{5}$$

4.2 Web Service Roll-back Mechanism

The Web services transaction mechanisms offer a significant mean to ensure the reliability of the service composition. They offer an efficient way to prevent resources being locked for a long time and improve the degree of concurrency between transactions Laura et al. (2003). The introduction of compensation transaction makes it possible to undo committed transactions. With an event-driven mechanism, compensation operations will only be performed when operations that affect the data or status of application system are concerned, including: (1) database modification operations (such as insert, update, delete, etc.), (2) coordination messages (such as begin, commit and abort, etc.); (3) user-defined critical points, etc. Corresponding compensation events are produced dynamically during execution of services.

In this paper, we employ a third-party Coordination Processing System (CPS) based on Web services transactions to assist in solving service conflict dynamically. During the execution of a composite service plan, each service submits a *Committed* message to the CPS within time frame t after the successful execution. CPS packages all of the compensation operations produced by the service into a compensating transaction and stores them in a database. If a service submits a *Failed* message to CPS or failed to submit a *Committed* message within time frame t, the service would be regarded as having failed, the service will self-roll-back the executed operation. Throughout the execution cycle of the composite service, the system can utilize the *Cancel* message to cancel a service. A compensation transaction of a service can be invoked to return the service to a previous state.

However, rolling back a previously completed transaction is potentially expensive Benchaphon et al. (2004). In order to get the optimal roll-back compensate strategy, calculating roll-back compensation cost is necessary. The transaction QoS attributes are defined as follows: execution fee of transaction t_i is denoted as $q_{pay}t_i$; execution duration of transaction is denoted as $q_{time}t_i$; and negative effect of transaction t_i is denoted as $q_{ne}t_i$. These attributes are normalised according to equation (6), where q_{max} and q_{min} respectively represent the maximum and minimum values of all the dimensions, i.e. execution fee, execution duration and negative effect. Afterwards, the character of normalised criteria is

Algorithm 1 Calculation of utility value

Input: SM: Service Model,Q: quality criteria value SR: Successful Change Rate, W: weight of criteria
Output: u: Utility value of the service plan

function Scale($criteriaVector$)
  if q_i is positive **then**
    for all $q_i \in criteriaVector$ **do**
      if $1/n \sum_{i=1}^{n} q_{i,j} \neq 0$ **then**
        $q_i' \leftarrow \frac{q_{i,j}}{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}$
      else
        $q_i' \leftarrow l_j$
      end if
    end for
  end if
  if q_i is negative **then**
    for all $q_i \in criteriaVector$ **do**
      if $q_{i,j} \neq 0$ and $\frac{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}{q_{i,j}} \leq l_j$ **then**
        $q_i' \leftarrow \frac{\frac{1}{n}\sum_{i=1}^{n} q_{i,j}}{q_{i,j}}$
      else
        $q_i' \leftarrow l_j$
      end if
    end for
  end if
end function
$q \leftarrow \sum_1^n w_i * q_i$
scale(Q)
scale(RS)
$U \leftarrow Q + RS$
return U

$$r_x(t_i) = \frac{q_x(t_i) - q_x^{min}}{q_x^{max} - q_x^{min}} \text{ where } x \in \{\text{pay, time, negtiveeffect}\} \tag{6}$$

$$Cost_{exec}(t_i) = w(p) * r_{pay}(t_i) + w(t) * r_{time}(t_i) + w(n) * r_{ne}(t_i). \quad (7)$$

$r_x(t_i) \in [0, 1]$. A lower value of $r_x(t_i)$ indicates a higher priority of transaction t_i. The calculation of roll-back compensation cost is based on multiple criteria decision making and the equation is

4.3 A Bi-Objective Model for Service Property Conflicts

Applying the bi-objective optimization Olivier et al. (2004) to Web services, we propose two objective functions $v_i^1 = f_1(c_1, c_2, p)$, $v_i^2 = f_2(c_1, c_2, p)$. v_i^1 is the profit gained from the new execution plan, v_i^2 is the compensation profit of roll-back, which is the opposite of the compensation cost. The strategy space of a conflicting service is denoted as $C = \{hold, drop\}$. For a service ws_i the strategy is denoted as $c_1 \in C$, where $hold$ means service ws_i will continue to be used, and $drop$ means service ws_i will have to be rolled back and compensated. Similarly, for ws_{i+1} we have $c_2 \in C_2$, with again $hold$ meaning that the service ws_{i+1} is retained, and $drop$ means service ws_{i+1} is compensated. Further recall that $M = \{p_1, p_2, \ldots, p_i, \ldots, p_n\}$ is the set of all plans. $p_i \in M$ denotes a composite service plan in the service model M, and the total number of execution plans is n. The constraint condition of the feature service conflict bi-objective model is $v_I^1 \geq \lambda_{plan}, v_I^2 \geq \lambda_{compensate}$. It means that the profit of the new execution plan should be equal or greater than threshold λ_{plan}, and the compensation profit should be equal or greater than $\lambda_{compensate}$. The values of λ_{plan} and $\lambda_{compensate}$ are specified by the user.

In order to get the optimal resolution strategy for conflicts, we aim to maximise the value of the two objective functions. Therefore, the bi-objective model established in the paper is

$$\begin{cases} max[v_I^1, v_I^2] \\ v_I^1 \geq \lambda_{plan}, v_I^2 \geq \lambda_{compensate} \end{cases}$$

A fundamental characteristic of a bi-objective optimization problem is the conflict between the two objectives. The result of the bi-objective problem is not a unique solution, but a set of Pareto solutions under constraints. In this paper, the solution of the bi-objective model is a trade-off between user expected utility of new plan and profit of roll-back.

$v_i^1 = f_1(c_1, c_2, p)$ denotes the revenue gained when service ws_i adopts strategy c_1, service ws_{i+1} adopts strategy c_2, and p is the new plan to be executed. Suppose a conflict between ws_i and ws_{i+1} is detected during execution of the service plan $p_{executing}$, a resolving strategy essentially includes the choice operation of ws_i and ws_{i+1} and an executable service plan p_{backup}.

When the choice operation of ws_i and ws_{i+1} are c_1 and c_2 respectively, if the state of ws_i and ws_{i+1} in another service plan is in accordance with c_1 and c_2 , then we define the service plan as backup service plan, denoted by p_{backup}. The set of all backup service plans is defined as backup service plan set, denoted by $P_{backup} = \{p_{backup_1}, p_{backup_2}, \cdots, p_{backup_n}\}$. One special case is when the choice strategy of ws_i and ws_{i+1} is *{hold,hold}*, where the backup service plan set will be empty: $P_{backup} = \emptyset$.

Therefore, the formula to calculate revenue of the new service plan is:

$$v_I^1(c_1, c_2, p) = \begin{cases} U, & \text{if } p \in P_{backup} \\ L, & \text{others} \end{cases} \quad (8)$$

Equation (8) means that when service plan p is an executable plan; that is, the state of ws_i and ws_{i+1} is in accordance with the choice of c_1 and c_2, the revenue is equal to user expected utility. Otherwise, the revenue of the new service plan is L, where L is much less than the minimum expected utility value of all the plans; i.e., $L \ll min\{u(p_1), u(p_2), \cdots, u(p_n)\}$.

Similarly, $v_i^2 = f_1(c_1, c_2, p)$ represents the roll-back revenue gained when service ws_i adopts strategy c_1, service ws_{i+1} adopts strategy c_2, and p is the new plan to be executed. When the strategy (c_1, c_2, p) is adopted, some executed services need to be rolled back. The roll-back set $RBset$ is obtained as follows. If $p \in P_{backup}$, then $RBset$ contains all the heterogeneous service nodes between the service from the beginning to the right executing service. Otherwise, if $p \notin P_{backup}$, then $Rset = \emptyset$. The roll-back revenue V_I^2 is the negative value of the sum of the roll-back cost of all the services in $RBset$, as shown in equation (9), where $rb(ws_i)$ is the roll-back cost of ws_i that can be calculated by using the method provided in subsection 4.2.

$$V_I^2(c_1, C_2, p) = -\sum_{Rset} rb(ws_i) \tag{9}$$

Following the efficient solution defined in Li et al. (1998), we designed Algorithm 2 to obtain Pareto solutions to the bi-objective optimization problem.

Algorithm 2 ParetoSet Solution

Input: SM: Service Model,U:utility of every plan, $Cost$:roll-back cost
Output: $ParetoSet$

if $holdDilemma$ is $\{hold, hold\}$ **then**
  $P_{backup} \leftarrow \emptyset$
else
  $P_{backup} \leftarrow \{p_i\}$
end if
if $p_i \in P_{backup}$ **then**
  $v_I^1 \leftarrow u$
else
  $v_I^1 \leftarrow L$
end if
$v_I^2 \leftarrow -\sum_{s \in RbSet} Cost(s_i)$
if $v_I^1 \geq \lambda_{plan}$ and $v_I^2 \geq \lambda_{compensate}$ **then**
  if $Strategy_i$ is not controlled by any other strategies **then**
    $ParetoSet \leftarrow Strategy_i$
  end if
end if
return $ParetoSet$

The efficient solution set of the bi-objective model reflects the relationship between the utility of new service plan and the revenue of roll-back. Each point in set R_{pa}^* represents an independent solution that is not controlled by any other solution. That is, all the solutions in R_{pa}^* meet the demands of resolving the conflict. Algorithm 3 depicts how to select the optimal solution from R_{pa}^*.

In this paper, we assume that a larger value for the two objectives is better. That is, we hope the distance between the solution and the *positive ideal point* is as small as possible,

Algorithm 3 Optimal Solution

Input: $ParetoSet$
Output: $OptimalStrategy$
1: $v_I^{1+} \leftarrow max\{v_{Ii}^1\}$
2: $v_I^{2+} \leftarrow max\{v_{Ii}^2\}$
3: $v_I^{1-} \leftarrow min\{v_{Ii}^1\}$
4: $v_I^{2-} \leftarrow min\{v_{Ii}^2\}$
5: $v^+ \leftarrow (v_I^{1+}, v_I^{2+})$
6: **for all** $s_i \in ParetoSet$ **do**
7: $\quad R_i \leftarrow \sqrt{(v_{Ii}^1 - v_I^{1+})^2 + (v_{Ii}^2 - v_I^{2+})^2}$
8: $\quad r_i \leftarrow \sqrt{(v_{Ii}^1 - v_I^{1-})^2 + (v_{Ii}^2 - v_I^{2-})^2}$
9: $\quad \xi_i \leftarrow R_i/R_i + r_i$
10: **end for**
11: $rank(\xi_i)$
12: **return** ξ

and the distance to *negative ideal point* simultaneously as large as possible. The *positive ideal point* means the two objective values are the maximal, the *negative ideal point* means the two objective values are the minimal. We mark positive ideal points as $v^+ = (v_I^{1+}, v_I^{2+})$ and *negative ideal points* as $v^- = (v_I^{1-}, v_I^{2-})$,where $v_I^{1+} = max\{v_{Ii}^1\}$, $v_I^{2+} = max\{v_{Ii}^2\}$, $v_I^{1-} = min\{v_{Ii}^1\}$, $v_I^{2-} = min\{v_{Ii}^2\}$. The distance to the positive ideal point is denoted by $R_i = \sqrt{(v_{Ii}^1 - v_I^{1+})^2 + (v_{Ii}^2 - v_I^{2+})^2}$ and distance to the negative ideal point is denoted by $r_i = \sqrt{(v_{Ii}^1 - v_I^{1-})^2 + (v_{Ii}^2 - v_I^{2-})^2}$; Further, the relative distance to the ideal point is denoted by $\xi_i = R_i/R_i + r_i$.

Finally, we rank the resolutions. Larger ξ_i represent a larger distance to the ideal point. And the highest ξ_i is the optimal solution. Other solutions can be used as alternatives, which can be selected when further accidents occur in the current solution. If there are several solutions with maximal distance, one of them is selected randomly. If no solution satisfies the user constraints for a given task, an execution exception will be raised and the system will ask the user to relax their constraints.

5 Performance Evaluation

In order to evaluate performance of the proposed resolution approach for run-time service property conflicts, we implemented the proposed scheme in an on-line shopping application system and conducted experiments using the prototype system. The business process for the online shopping comprises the sequence of tasks – accessing sale platform, purchasing in shops, paying through a bank account and payment partner, and delivery via express logistic. Multiple services are available for each task of the workflow. The service model that contains all possible service plans for realizing the workflow is shown in Figure 5.

In the service model, there are totally six service plans $(p_1, p_2, p_3, p_4, p_5, p_6)$. According to the user's preference, which is specified by the weights for QoS attributes provided the user, the expected utility of every service plan is calculated by using formulas (1) to (5), and shown in Figure 5 (the values $P_i(x)$ in the upper left). We can see that p_4 has the maximum

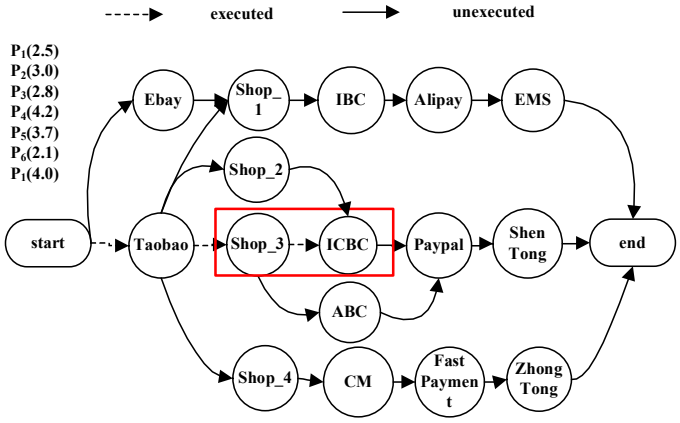


Figure 5: Composition services model of E-commerce

Table 5 Solution of trade-off strategy and backup plans

Holding Dilemma	Executable Plan Set
{hold,hold}	$P_{backup} = \emptyset$
{hold,drop}	$P_{backup} = p_5$
{drop,hold}	$P_{backup} = p_3$
{drop,drop}	$P_{backup} = \{p_1, p_2, p_6\}$

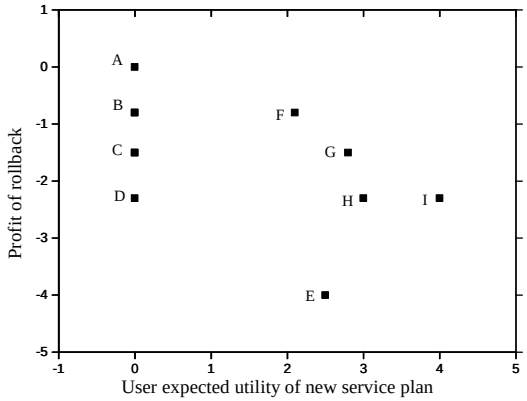
Table 6 new plan Utility and rollback cost of different strategies

rollbackcost / newplan / strategy	P1	P2	P3	P4	p5	p6
hold,hold	L/0	L/0	L/0	L/0	L/0	L/0
hold,drop	L/-0.8	L/-0.8	L/-0.8	L/-0.8	2.1/-0.8	L/-0.8
drop,hold	L/-0.8	L/-0.8	L/-0.8	L/-0.8	2.1/-0.8	L/-0.8
drop,drop	2.5/-4.0	3.0/-2.3	L/ -2.3	L/-2.3	L/-2.3	4.0/-2.3

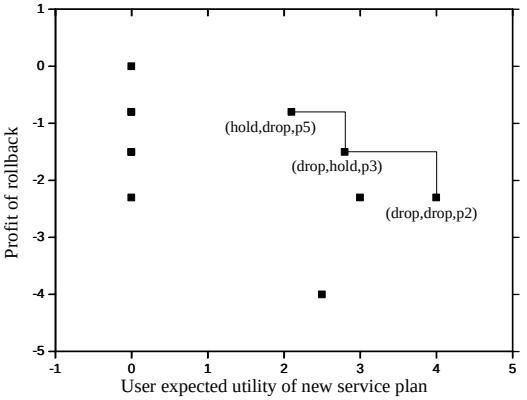
expected utility value (4.2) thus is the best plan for meeting user requirements. Therefore, p_4 is chosen for execution. However, during execution, a property conflict between the service *Shop_3* and service *ICBC* (as shown by the boxed area in Figure 5) is detected and needs to be resolved.

Applying the method we developed in the previous section, holding strategies of services *Shop_3* and *ICBC* and backup service plans can be computed. The results are shown in Table 5. We assume that the cost of rolling back service Taobao is 1.7 , that of service Shop_3 is 1.5 and that of service ICBC is 0.8. The thresholds are $\lambda_{compensate} = -3.5, \lambda_{plan} = 0.5$.

Table 6 shows the related data including the expected utility value and roll-back cost of different strategies.



(a) Range set of Bi-Objective Optimal



(b) Efficient points set of Bi-Objective Optimal

Figure 6: Range set and efficient points set of Bi-Objective Optimal.

Figure 6a shows the image set of bi-objective optimization. In the paper, the two objectives are the maximum value. Li et al. (1998) concludes that efficient point lies on the boundary of image set $F(R)$ and when the boundary AB of the image set in the second quadrant is monotonically decreasing then AB is efficient point set. As shown in Figure 6b, the image set boundary in the second quadrant is $R^*_{pa} = \{A, F, G, I\}$. But the user expected utility of point A is 0, less than threshold $\lambda = 0.5$. Therefore, the effective points set is $R^*_{pa} = \{A, F, G\}$. Here $L = 0$.

Figure 6b shows that R^*_{pa} *={(hold,drop,p5), (drop,hold,p3), (drop,drop,p2)}*. In this case,

$v_I^{1+} = max(2.1, 2.8, 3.0) = 3.0$,
$v_I^{2+} = max(-0.8, -1.5, -2.3) = -0.8$,
$v_I^{1-} = min(2.1, 2.8, 3.0) = -2.1$,
$v_I^{2-} = min(-0.8, -1.5, -2.3) = -2.3$.
The distances between different solutions and the ideal point are $\xi_1 = 0.375, \xi_2 = 0.406, \xi_3 = 0.625$ respectively. So the optimal solution is $(drop, drop, p2)$. Therefore we obtain the optimal strategy for resolving the conflict is to roll-back service $Shop\_3$ and $ICBC$, and execute service plan $p2$, which can successfully complete the service for meeting the initial user requirements.

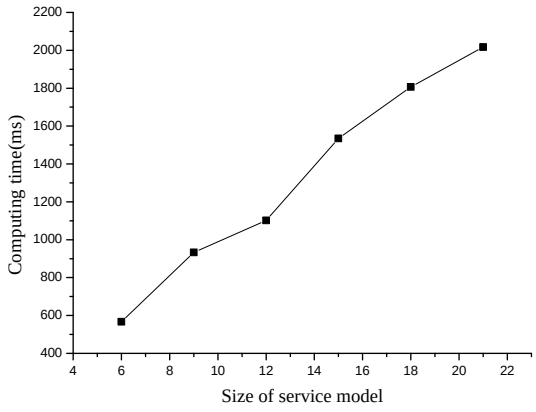


Figure 7: Average run time of the algorithm

Since the proposed scheme is for resolving run-time conflicts of service properties, the response time of the resolution algorithm is an important aspect of performance evaluation for the scheme. We conducted experiments with different sizes of service models with different number of feasible service plans to measure the response time for conflict resolution. We repeated the experiments for each service model size for 10 times and calculated the average resolution response time as the measurement result. The obtained results of average response time for different service model sizes are plotted in Figure 7.

we can see from Figure 7 that the response time of run-time conflict resolution increases with the service model size. This is because a larger set of feasible service plans introduces a large solution space in which the bi-objective optimization algorithm must search and rank Pareto solutions in order to obtain the optimal roll-back strategy and alternative service plan. This figure also indicates that for the largest size of service model tested in our experiments, which has 21 possible service plans, the resolution response time is about 2 seconds. For typically e-commerce service systems used in practice, the total number of feasible service plans will be within this range. A response time of less than 2 seconds is prompt enough for run-time resolution of service property conflicts.

We also noticed from Figure 7 that the response time increases with service model size linearly. This implies that the proposed resolution scheme could have a long response time for complex service provisioning systems with a large number of feasible service plans. Therefore, we recommend application of the proposed scheme to small/medium size

service composition systems, and plan to further improve the efficiency and scalability of the resolution algorithm for large systems in our future work.

6 Related Work

Since the feature interaction problem in the domain of Web services was first proposed in the International feature interaction workshop in 2003, the problem has attracted a number of studies (Gang et al., 2008; Amyot et al., 2003; Michael et al., 2004, 2007; Luo et al., 2010). Studies in feature interaction focused on the requirements analysis phase. But in the dynamic Web services composition, conflicts often occur during service execution; thus cannot be completely avoided in the design phase. Therefore, in order to improve the efficiency of Web services composition, on-line solution of service property conflicts is needed. Dave et al. (1998) presented an approach to automated detection and resolution of feature interactions during runtime using techniques borrowed from transactions processing theory based on a feature interaction manager, and proposed a simple on-line solution based on transaction roll-back mechanism. Zhang et al. (2007b) presented an immune-inspired on-line detection system for WSFI problem. Xu et al. (2011, 2010) adopt the method of situation calculus to achieve a dynamic detection of feature interaction, while Chen et al. (2010) proposed a multi-solutions service conflict resolver based on a Markov decision model. They also presented a case study to analyse, but it does not make use of actual combination services to conduct experimental runtime validation.

In this paper, we established a user-centred bi-objective optimization model to obtain a strategy to dynamically resolve property conflicts in service compositions. Our method considers user's revenue as prime aim by obtaining the optimal strategy for process continuation. By considering multiple QoS attributes in the model, our method achieves the advantage of being able to guarantee optimal QoS performance while obtaining an alternative service plan for resolving a service property conflict.

7 Conclusion

In the field of Web services, due to the large number of available services developed and deployed independently by various providers, property conflicts between services become a serious obstacle for service composition to meet users' QoS requirements. Some service property conflicts occur only during execution of composite services; thus must be resolved online at run-time. In this paper, we tackled the problem of run-time resolution of service property conflicts using a bi-objective optimization method. We first developed a user revenue-based utility model for measuring QoS performance and successful rate of possible service plans; and provided a method for calculating compensation cost for rolling back executed services. Then we proposed an optimization model for simultaneously maximizing service utility and minimizing roll-back cost when resolving service property conflicts. We also design algorithms for solving the bi-objective optimization problem to get a Pareto efficient solution set and sorting the set to obtain the optimal solution. The obtained solution provides the optimal roll back strategy and alternative service plan that can complete the service execution for meeting the user's requirements. We implemented the proposed resolution scheme in a prototype system of online shopping applications and conducted experiments to evaluate performance of the scheme. Obtained results verified effectiveness

of the scheme and indicated that its response time is reasonable for run-time resolution of property conflicts in service compositions for typical small to medium size e-commerce applications. As future work we plan to further enhance efficiency of the resolution scheme for large service composition systems and integrate both detection and resolution of service property conflicts into one unified run-time service management system.

Acknowledgement

The paper is fully supported by a grant from the Fundamental Research Funds for the Central Universities (Project No. 13CX06009A and No. 14CX06007A).

References

D Amyot and L Logrippo. Feature interactions in web services. *Feature Interactions in Telecommunications and Software Systems VII*, page 149, 2003.

Bocchi, Cosimo Laneve, and Gianluigi Zavattaro. A calculus for long-running transactions. In *Formal Methods for Open Object-Based Distributed Systems*, pages 124–138. Springer, 2003.

de Weck and Il Yong Kim. Adaptive weighted sum method for bi-objective optimization. In *Proceedings of the 45th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference*, 2004.

Esfandiari and Vladimir Tosic. Requirements for web service composition management. In *Proceedings of the 11th HPOVUA Workshop*, 2004.

Gang Huang, Xuanzhe Liu, and Hong Mei. Online approach to feature interaction problems in middleware based system. *Science in China Series F: Information Sciences*, 51(3):225–239, 2008.

Chen Kun. Research on automated web service composition based on AI planning. Master's thesis, China University of Petroleum, 2010.

Benchaphon Limthanmaphon and Yanchun Zhang. Web service composition transaction management. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 171–179. Australian Computer Society, Inc., 2004.

Dave Marples and Evan H Magill. The use of rollback to prevent incorrect operation of features in intelligent network based systems. In *FIW*, pages 115–134, 1998.

Justin O'Sullivan, David Edmond, and Arthur Ter Hofstede. What's in a service? *Distributed and Parallel Databases*, 12(2-3):117–133, 2002.

Michael Weiss and Babak Esfandiari. On feature interactions among web services. In *Web Services, 2004. Proceedings. IEEE International Conference on*, pages 88–95. IEEE, 2004.

Michael Weiss, Babak Esfandiari, and Yun Luo. Towards a classification of web service feature interactions. *Computer Networks*, 51(2):359–381, 2007.

Jiuyun Xu, Kun Chen, Youxiang Duan, and Stephan Reiff-Marganiec. Modeling business process of web services with an extended strips operations to detection feature interaction problems runtime. In *Web Services (ICWS), 2011 IEEE International Conference on*, pages 516–523. IEEE, 2011.

Jiuyun Xu, Wengong Yu, Kun Chen, and Stephan Reiff-Marganiec. Web services feature interaction detection based on situation calculus. In *Services (SERVICES-1), 2010 6th World Congress on*, pages 213–220. IEEE, 2010.

Luo Xiangyu, Tan Zheng, and Dong Rongsheng. Atutomated detaction method for web service feature interaction. *Computer Science*, 37(012):106–109, 2010.

Li Yisheng. The effective set points of bi-objective optimization and the arch dam bi-objective optimization. *water power*, 11:10–13, 1998.

Jianyin Zhang, Sen Su, and Fangchun Yang. Detecting race conditions in web services. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, pages 184–184. IEEE, 2006.

Jian-yin Zhang, Fang-chun Yang, and Sen Su. Detecting feature interactions in web services with model checking techniques. *The Journal of China Universities of Posts and Telecommunications*, 14(3):108–112, 2007.

Jianyin Zhang, Fangchun Yang, Kai Shuang, and Sen Su. Immune-inspired online method for service interactions detection. In *SOFSEM 2007: Theory and Practice of Computer Science*, pages 808–818. Springer, 2007.