Journal of Zhejiang University-SCIENCE C (Computers & Electronics)
ISSN 1869-1951 (Print); ISSN 1869-196X (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

A self-optimizing QoS-aware service composition approach in a context sensitive environment

Yuan-hong SHEN, Xiao-hu YANG‡

(*School of Computer Science and Technology, Zhejiang University, Hangzhou 310027, China*)

E-mail: {yhshen, yangxh}@zju.edu.cn

Received Feb. 21, 2010; Revision accepted Sept. 10, 2010; Crosschecked Jan. 31, 2011

Abstract: QoS-aware service composition is aimed to maximize the global QoS of a composite service when selecting candidate services. In a context sensitive service execution environment in pervasive computing, the context information for service composition is not static: device, policy, and user constraints, and QoS requirements may change, new services may be deployed, old ones withdrawn, or existing ones change their QoS parameters. This results in the current service composition plan failing or its QoS degrading from the optimum. In this paper, a runtime self-optimizing service composition framework is proposed. An implementation of a prototype for this framework is presented, addressing the issues of reducing extra delay while increasing global QoS in service composition in a dynamic context environment. Three service re-plan algorithms are compared that can be used in dynamic context environment, i.e., minimal-conflict hill-climbing repair genetic algorithm (MCHC-repair GA), an improved penalty-based GA, and our multi-population conflicts sorted repair genetic algorithm (MP-CS-repair GA), as well as three kinds of service composition mechanisms—with backup, without backup, and our context-aware service re-selection mechanisms. The results show that our MP-CS-repair GA and context-aware service re-selection method can reduce more extra delay while acquiring a higher global QoS for the composite service in a context sensitive environment. This context-aware service re-selection mechanism also shows some adaptability to different context change frequencies and user requirements for reducing computation cost in the self-optimizing process.

Key words: Service composition, Quality of service (QoS), Context, Repair genetic algorithm, Self-optimizing
doi:10.1631/jzus.C1000031 **Document code:** A **CLC number:** TP31

1 Introduction

Web services present a promising technology to compose complex service applications from individual (atomic) services (Yu and Lin, 2005a). Service composition is aimed to find a candidate service for each abstract service (task) in a composite from a distributed environment to realize a composite service (Ye and Mounla, 2008). As many service providers might provide multiple candidate services with the same or overlapping functionality, the quality of service (QoS) of these candidate providers, such as cost, execution time, and availability, can be used to distinguish these service providers (Ye and Mounla, 2008).

In a pervasive computing environment, there exist device, policy, and other constraints and conditions called context from service providers and requesters that impose the QoS relationship and affect the usability and QoS of candidate services for composite services. For example, some candidate services are not compatible with each other in the same composite service; the QoS values of one service may depend on the other service claimed by service providers (Zhou *et al*., 2008). Device constraints are also prevalent and should be met, considering that the two atomic services interacting with each other in a composite service are in two different devices. What is more, the context information for service composition is not static; device, policy, and user constraints, and QoS requirements may change, new services may be deployed, old ones withdrawn,

‡ Corresponding author

or existing ones change their QoS parameters, which results in the global QoS of the running service composition plan degrading from the optimum. In a dynamic context environment, the service composition must have a self-optimizing ability, which means that service composition is throughout the process when the composite service is running, and the composite service can heal by itself if any execution environment changes or its QoS degrades a lot, in order to complete its execution successfully, meeting a high QoS requirement and different constraints.

The easiest solution to this problem is to re-select the services once a context change occurs, which is time consuming and thus non-feasible. And, the execution of the composite service will be interrupted frequently (Dai *et al*., 2009). Canfora *et al*. (2005b) proposed another re-selection in the execution of the composite service, where the re-selection is triggered as soon as the actual QoS deviates from the initial estimates. The execution of the composite service will not continue until the re-selection completes; however, this approach can be used only for runtime-unaware applications (Dai *et al*., 2009). To minimize the extra delay caused by re-plan, Yu and Lin (2005b) and Girish *et al*. (2006) proposed the solution of backing up a replacement service for each task. When one task in the composite server insures a failure, the composite service can easily switch to a replacement, avoiding an extra delay. Recently Dai *et al*. (2009) improved the availability of replacement composite services by performance prediction, which enables the composite service to heal by itself from a failure as quickly as possible and minimizes the number of re-plans. The performance prediction was focused on the transmission speed in supporting modeling reliability of services for better QoS and higher reliability of replacement of composite services.

However, in a dynamic pervasive computing environment, more QoS attributes for service composition exist and more context information, including QoS of all candidate services for each task and device and policy constraints for the unexecuted task, may change, which results in a non-feasible pre-selected service execution plan or the global QoS degrading a lot from the optimum. Since there are more complicated factors influencing the performance of a composite service, it is difficult to obtain a service reliability model through predicting each kind of QoS attribute and each type of context information for each unexecuted task.

With all these problems in mind, we present a new self-optimizing QoS-aware service composition approach in a context sensitive environment. To make things simpler in predicting the QoS degraded unexecuted selection plan for replacement, we predict the global QoS of unexecuted workflow by analysis of periodic context similarity between the current context information and the historical one for the current execution plan. If the context similarity is below a threshold, the global QoS of the current service composition plan is more likely to degrade a lot from the current optimum. Thus, a service re-selection process is triggered at the runtime of the composite service for an unexecuted workflow to optimize the global QoS of unexecuted tasks. The re-selection process will not affect the execution of the composite service because it is performed while the composite service is running. In addition, this context-aware service re-selection mechanism also shows adaptability to different context change frequencies and user requirements for less computation cost by reducing the number of re-selection times.

In another aspect, to decrease the extra delay caused by re-plan when the execution of the composite service fails, we back up the replacement service for each task. However, if the execution of the current task in the composite service and the execution of its backup both fail, the whole composite service suspends and the selection process is triggered for all unexecuted tasks to find new candidate services to maximize global QoS, which might be time consuming because it is an NP problem. Recently genetic algorithms (GAs) were used for the re-plan process (Gao *et al*., 2007; Canfora *et al*., 2008; Gong *et al*., 2008). We choose GA for service selection/re-selection/re-plan mainly because its ability in optimizing the global QoS in the context dynamic environment with multiple and variable constraints. To shorten this re-plan process while achieving a high global QoS for the whole workflow, we propose a new multi-population conflicts sorted (MP-CS) repair GA to quickly switch the current service execution plan that failed to a new successful one with optimized global QoS, meeting all the current context constraints.

The contribution of this paper includes: (1) A new self-optimizing framework as well as its prototype implementation in the experiments is proposed for QoS-aware service composition in a context sensitive environment; (2) A context-aware service re-selection mechanism is presented by context similarity analysis to increase the global QoS of composite services; (3) A new MP-CS repair GA is proposed to decrease the extra delay caused by re-plan when the execution of one task and the execution of its backup in the composite service both fail.

2 Background

2.1 Context-free QoS-aware service composition

Zhou *et al*. (2008) reported that a context-free QoS model for a business workflow P is described as follows: P is represented as a set of abstract services AS_i; each AS_i has m CSs as its candidate services, $AS_i=\{CS_{ij}|0<i\leq n, 1\leq j\leq m\}$; for each CS_{ij}, $CS_{ij}.Q$ is a vector describing its quality metrics, and $CS_{ij}.Q=[c, l, a, r]$, where c is the cost, l the latency, a the availability, and r the reliability.

We use multiple criteria decision making and simple additive weighting techniques (Ye and Mounla, 2008) to calculate the global QoS of a composite service. Define $V_{\max}$ and $V_{\min}$ as the maximum and the minimum values of a global QoS property of a composite service, respectively, and V as an execution global QoS property. $V_{\max}$ and $V_{\min}$ are calculated by choosing each candidate service with the maximum and minimum QoS property values while V is calculated using the QoS prosperity value of each executed service. The global QoS calculation methods differ according to the different workflow (composite service) patterns, as shown in Table 1 (Canfora *et al*., 2004).

Table 1 Common parameters used for different workflow patterns

Composite QoS	Sequence	Fork	Choice	Loop
c	$\sum_{i=1}^{n} c_i$	$\sum_{i=1}^{n} c_i$	$\sum_{i=1}^{n} p_i c_i$	nc
l	$\sum_{i=1}^{n} l_i$	$\max_{i=1,2,\ldots,n} l_i$	$\sum_{i=1}^{n} p_i l_i$	nl
a	$\prod_{i=1}^{n} a_i$	$\prod_{i=1}^{n} a_i$	$\sum_{i=1}^{n} p_i a_i$	a^n
r	$\prod_{i=1}^{n} r_i$	$\prod_{i=1}^{n} r_i$	$\sum_{i=1}^{n} p_i r_i$	r^n

c: cost; l: latency; a: availability; r: reliability. n: number of atomic services; p: probability of execution for each branch

Then the cost and latency values are normalized using Eq. (1) and availability and reliability values are normalized using Eq. (2):

$$\mathrm{SV}=\frac{V_{\max}-V}{V_{\max}-V_{\min}}, \tag{1}$$

$$\mathrm{SV}=\frac{V-V_{\min}}{V_{\max}-V_{\min}}. \tag{2}$$

Eq. (3) describes the calculation method of the QoS score of a service:

$$\mathrm{Score}=\sum_{i\in \mathrm{PS}} \mathrm{SV}_i \cdot W_i, \tag{3}$$

where PS represents the set of QoS properties of a service, W_i the weight assigned to property i, and SV_i the scaled value of property i (Ye and Mounla, 2008). In context-free QoS-aware service composition scenarios, the aim is to find an execution plan that can maximize the QoS score of a composite service, which is to maximize the result of an object function in Eq. (3).

2.2 QoS related context

Context encompasses all the information about the client of a Web service that may be used by the Web service to adjust execution and output delivery so that the client can benefit from a customized and personalized behavior (Keidl and Kemper, 2004). For the QoS driven service composition problem, the context information that influences the selection of candidate service is collected and we name it 'QoS related context', which can be classified into four types:

1. Service context (Tang *et al*., 2008): QoS information and device requirements of candidate services that meet functional and other requirements in QoS-aware service composition. Service context= $\{CS_{ij}.Q, CS_{ij}.\mathrm{value}, B_{\mathrm{required}}(CS_{ij}, CS_{i'j'})|1\leq i, i'\leq n, 1\leq j, j'\leq m\}$, where n is the number of tasks in P, m is the number of candidate services for each AS_i, $CS_{ij}.\mathrm{value}$ represents the inner device resource requirements, $B_{\mathrm{required}}(x, y)$ represents the minimal required bandwidth between services x and y if x and y require interaction with each other. Service context is included in the device context, policy context, and so on, to represent complicated constraints and conditions influencing service composition.

2. Device context (Tang *et al*., 2008): the inner device resource constraints and resource constraints between different devices (candidate services are deployed on the devices):

(1) Inner device resource constraints (Tang *et al*., 2008): if k candidate services CS_1, CS_2, …, CS_k are deployed on device d_s for running in a composite service, then the total resource requirements of k candidate services (resource requirement of CS_i is represented as CS_i.value) will not exceed the total available resources of device d_s (which is represented as d_s.value); that is, $\sum_{i=1}^{k} CS_i.\text{value} \leq d_s.\text{value}$. (C1)

(2) Resource constraints between devices (Tang *et al*., 2008): if service CS_i is deployed on device d_l, CS_j is deployed on device d_p, and interaction is required between CS_i and CS_j, then the bandwidth requirements should be met; that is, $B(d_l, d_p) \geq B_{\text{r quired}}(CS_i, CS_j)$, where $B(x, y)$ represents the bandwidth between devices x and y. (C2)

3. Policy context (Yu *et al*., 2008; Zhou *et al*., 2008): the business rules that influence the selection of candidate services for a given time. There are many types of policy context, including:

(1) Exclusive constraints (Zhou *et al*., 2008): the candidates who can or cannot be selected in some situations. For example, if you select CS_{31} from service provider *A*, you must select CS_{42} from service provider *B*. (C3)

(2) Single constraints (Zhou *et al*., 2008): QoS restrictions on a single selected service. For example, the cost of the first task is lower than 100. (C4)

(3) Combined QoS constraints (Zhou *et al*., 2008): QoS restrictions on more than one selected service. For example, the total execution time of the composite service is less than 500 s. (C5)

(4) Combined QoS change (Zhou *et al*., 2008): the rules of QoS change of several CS candidates for selection. For example, if you select both CS_{21} and CS_{32}, you will obtain 30% discount on CS_{21} and 20% discount on CS_{32}. (C6)

4. User satisfaction enhancement (Zhou *et al*., 2008): the extra value enhanced by service providers. For example, whenever CS_{31} is bought from provider *A*, CS_{42} is free. (C7)

2.3 Genetic algorithm and repair genetic algorithm

GAs (Renders and Flasse, 1996) can be used for a global optimization problem. A search is performed by evolving a population of candidate solutions through the use of nondeterministic operators and by improving incrementally the individuals forming the population by mechanisms inspired from those of genetics (e.g., crossover and mutation). The process of GA includes population initialization and new offspring generations by applying selection, crossover, and mutation operators, repeated until the algorithm converges. There are many solutions to improving the efficiency of GA while avoiding local optima, e.g., the gene space balance strategy (GSBS) (Hu *et al*., 2007), the information entropy based selection strategy (Cui and Lin, 2004), roulette-wheel selection, and linear-rank selection, to keep the diversity of each generation and increase evolutionary efficiency.

For constrained optimization, two different constraint handling mechanisms can be incorporated into GAs. One is the penalty mechanism, that is, to give a penalty to non-feasible individuals (the individuals who violate the constraints) when evaluating the fitness of the reproduced individuals. The GA with a penalty mechanism is called penalty GA. When the constraint density is high, the fitness function will be complicated and should be designed carefully to handle constraints of different kinds, since it affects the effectiveness of the algorithm. Another approach is to use domain specific knowledge to repair the non-feasible individuals in the population such that all the individuals are always feasible. The GA with this repairing mechanism is called repair GA.

3 Self-optimizing QoS-aware service composition framework

3.1 Overview of the framework

In this study, we propose a self-optimizing QoS-aware service composition framework in a context sensitive environment to simulate real business scenarios in a pervasive computing environment. The framework includes mainly four different components: context manager, service selection/re-selection trigger, QoS-driven context-based service selector/re-selector, and composite service executor. Fig. 1a presents this framework and Figs. 1b–1d describe its components.

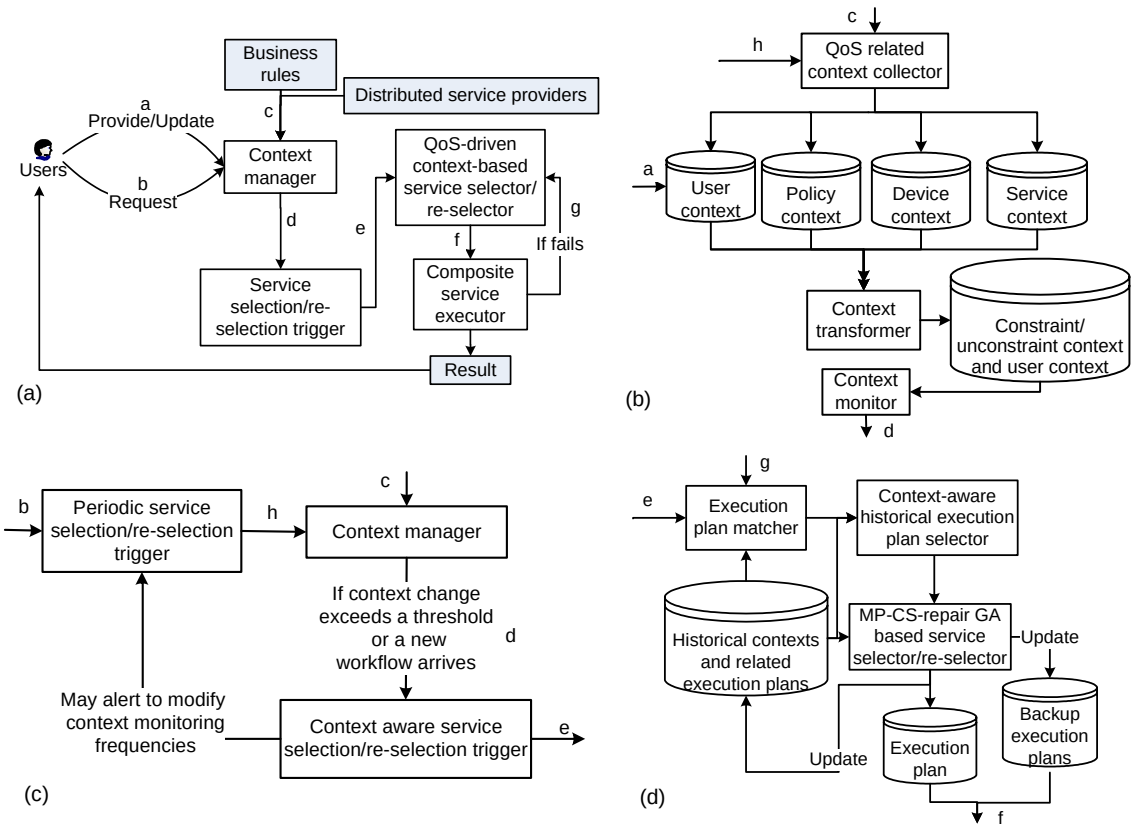

Fig. 1 Self-optimizing QoS-aware service composition framework in a context sensitive environment (a) and its components: context manager (b), service selection/re-selection trigger (c), and QoS-driven context based service selector/re-selector (d)

3.1.1 Context manager

The context manager includes three sub-modules:

The QoS related context collector collects QoS related context from business rules and distributed service providers when the periodic service selection/re-selection trigger is invoked.

The context transformer transfers the selected context into constraints, unconstraints, and user context (Section 3.2.1).

The context monitor performs context similarity analysis (Section 3.2.2) between the most recent context related to the current execution plan and the current context. If the context similarity is below a threshold, the service re-selection trigger will be invoked. The context monitor may also alert the periodic service selection/re-selection trigger to adjust the service re-selection frequency based on context change frequency. If the context change frequency is high, then the current execution plan may be invalid in a very short time. Thus, it is necessary to increase the service re-selection triggering frequency for the unexecuted tasks. If the context change frequency is low, it is not necessary to trigger the service re-selection process too frequently. Thus, the overall computation time spent in context collection and context similarity analysis can be reduced.

3.1.2 Service selection/re-selection trigger

The service selection/re-selection trigger includes two sub-modules:

The periodic service selection/re-selection trigger is invoked for collecting QoS related context. Service selection is triggered when a new workflow arrives; service re-selection is triggered periodically or when the composite service fails.

The context-aware service selection/re-selection trigger is invoked by the context manager to perform service selection/re-selection.

3.1.3 QoS-driven context based service selector/ re-selector

The QoS-driven context based service selector/re-selector includes three sub-modules: the execution plan matcher (Section 3.3), the context-aware historical execution plan selector (Section 3.4.2), and the MP-CS-repair GA based service selector/re-selector (Section 3.4).

The selector/re-selector also inserts and updates the historical contexts and related execution plans by a set of feasible execution plans of the service selector/re-selector results for the composite service. When a feasible service execution plan is acquired by the service selector/re-selector, the composite service executor is invoked. The selector/re-selector also prepares a backup service execution plan for the running composite service.

3.1.4 Composite service executor

The composite service executor executes the composite service and presents the results to users. If the current execution plan and its backup both fail, the QoS-driven context based service re-selector will be invoked to generate a new execution plan.

The key to our framework is context-aware service re-selection for the unexecuted workflow in a dynamic environment. Two technologies are important for facilitating our approach: one is context similarity analysis based on QoS related context transformation, and the other is an MP-CS repair GA to quickly optimize the current degraded service composition plan for an unexecuted workflow. Compared with the work of triggering the re-selection when the execution of a task fails or its QoS deviates a lot from prediction when executed, our approach uses the re-selection process when the global QoS of the unexecuted workflow is very likely to degrade from the current optimum based on context similarity analysis. Thus, the global QoS of the executed composite service is optimized. The re-selection process will not affect the execution of the composite service, since it is performed while the composite service is running. In another aspect, if the execution of the current task in the composite service and the execution of its backup both fail, the whole composite service suspends and the selection process is triggered for all unexecuted tasks to find new candidate services to maximize global QoS. This may be time consuming because it is an NP problem. Then we design an MP-CS repair GA to speed up the re-plan process, obtaining a high global QoS for unexecuted workflow. In addition, the service re-selection triggering frequency can be adjusted in the service composition process to adapt to context change frequency.

3.2 QoS related context transformation and analysis

3.2.1 QoS related context transformation

We introduce the following definitions for describing the transformed QoS related context based on the definitions in Section 2.1:

Definition 1 (Constraint context) Constraint context is represented by a set of constraint expressions acting on a set of subsets of E, i.e., $\mathrm{con}_1(\mathrm{ele}_1)$, $\mathrm{con}_2(\mathrm{ele}_2)$, …, $\mathrm{con}_i(\mathrm{ele}_i)$, $\mathrm{ele}_i \subseteq E$. $E=\{C(\mathrm{AS}_1), C(\mathrm{AS}_2), \ldots, C(\mathrm{AS}_m)\}$, where m is the number of ASs, and $C(\mathrm{AS}_i)$ represents a chosen candidate service number for AS_i in a composite service. Define M_{ij} as the chosen device number of CS_{ij} for AS_{ij}. Each constraint expression is defined according to some specific constraint formats. E.g., the transformed constraint expressions for contexts (C1) to (C5) are shown below:

$$\sum_{i=1}^{n}(M_{iC(\mathrm{AS}_i)} == j)\cdot C(\mathrm{AS}_i).\mathrm{value} \le d_j.\mathrm{value}, \quad (\mathrm{C1}^*)$$

$$(M_{iC(\mathrm{AS}_i)} == d_l) \wedge (M_{jC(\mathrm{AS}_j)} == d_p)$$
$$\Rightarrow B_{\mathrm{required}}(C(\mathrm{AS}_i), C(\mathrm{AS}_j)) \le B(d_l, d_p)$$
(interaction is required between $C(\mathrm{AS}_i)$ and $C(\mathrm{AS}_j)$), $(\mathrm{C2}^*)$

$$C(\mathrm{AS}_3)==1 \Rightarrow C(\mathrm{AS}_4)==2, \quad (\mathrm{C3}^*)$$

$$C(\mathrm{AS}_1).Q.c<100, \quad (\mathrm{C4}^*)$$

$$\sum_{i=1}^{n} C(\mathrm{AS}_i).Q.l < 500. \quad (\mathrm{C5}^*)$$

Definition 2 (Unconstraint context) Unconstraint context is represented by a set of unconstraint expressions describing QoS change for CS in a context sensitive environment. For example, the transformed unconstraint context for (C6) and (C7) are represented as follows:

$$\mathrm{CS}_{21}.Q.c=0.7\mathrm{CS}_{21}.Q.c \text{ if } C(\mathrm{AS}_2)==1,\ C(\mathrm{AS}_3)==2;$$
$$\mathrm{CS}_{32}.Q.c=0.8\mathrm{CS}_{32}.Q.c \text{ if } C(\mathrm{AS}_2)==1,\ C(\mathrm{AS}_3)==2. \quad (\mathrm{C6}^*)$$

$$\mathrm{CS}_{42}.Q.c=0 \text{ if } C(\mathrm{AS}_3)==1,\ C(\mathrm{AS}_4)==2. \quad (\mathrm{C7}^*)$$

Definition 3 (User context, Tang *et al*., 2008). User context is represented by users' non-functional

requirements (preferences) for service selection. In the service composition framework for optimizing global QoS, the non-functional requirement of the service is the global QoS value V as introduced in Section 2.1. $V\in\{V_c, V_l, V_a, V_r\}$ represents c, l, a, r for global QoS for a composite service. The user context is described by W_c, W_l, W_a, W_r (Section 2.1), corresponding to the weights of global c, l, a, and r.

3.2.2 Context similarity analysis

In this framework, context similarity analysis is performed after new QoS related context is transformed into constraint context, unconstraint context, and user context for service re-selection. If the context similarity for the unexecuted tasks between the new context and the historical context for the current execution plan is below a threshold, a service re-selection process will continue for the unexecuted tasks. The reason for this is based on the assumption that if context for the unexecuted abstract services changes more, the global QoS of the current execution plan will be more likely to degrade from the current optimum.

Constraint and unconstraint contexts, user context, and QoS information of candidate services all affect the result of QoS-aware service composition. Assume Context_A and Context_B represent different context information relating to a same set of abstract services for a same workflow. For example, if AS_1 and AS_2 require interaction for the workflow, then the context related to AS_1 and AS_2 includes (C1)–(C7), QoS information of candidate services of AS_1 and AS_2, and user context. Context similarity of Context_A and Context_B,, $\text{Sim}(\text{Context}_A, \text{Context}_B)$, consists of constraint and unconstraint context similarity $\text{Sim}(\text{Context}_A.c, \text{Context}_B.c)$, user context similarity $\text{Sim}(\text{Context}_A.u, \text{Context}_B.u)$, and QoS similarity of candidate services $\text{Sim}(\text{Context}_A.q, \text{Context}_B.q)$, each of which has a corresponding weight (W_c, W_u, or W_q):

$$\text{Sim}(\text{Context}_A, \text{Context}_B)=W_c\cdot\text{Sim}(\text{Context}_A.c, \text{Context}_B.c)+W_u\cdot\text{Sim}(\text{Context}_A.u, \text{Context}_B.u)+W_q\cdot\text{Sim}(\text{Context}_A.q, \text{Context}_B.q).$$

The range of Sim is [0, 1]; the higher the context similarity between Context_A and Context_B, the larger the $\text{Sim}(\text{Context}_A, \text{Context}_B)$.

$\text{Sim}(\text{Context}_A.c, \text{Context}_B.c)$=(overlapping context expression number between Context_A and Context_B)/(total context expression number−overlapping context expression number between Context_A and Context_B.

User context of Context_A is represented by W_{Ac}, W_{Al}, W_{Aa}, W_{Ar}, and user context of Context_B is represented by W_{Bc}, W_{Bl}, W_{Ba}, W_{Br}. Thus,

$$\text{Sim}(\text{Context}_A.u, \text{Context}_B.u)=(|W_{Ac}-W_{Bc}|+|W_{Al}-W_{Bl}|+|W_{Aa}-W_{Ba}|+|W_{Ar}-W_{Br}|)/2.$$

The QoS information of Context_A and Context_B is represented by $\text{CS}_{ij}.Q_A$ and $\text{CS}_{ij'}.Q_B$, respectively, the candidate service number related to AS_i in Context_A is represented by $\text{num}(C_A(\text{AS}_i))$, and the new candidate service number from Context_A to Context_B is represented by $\Delta\text{num}(C_B(\text{AS}_i))$, $1\le j\le \text{num}(C_A(\text{AS}_i))$, $1\le j'\le\text{num}(C_A(\text{AS}_i))+\Delta\text{num}(C_B(\text{AS}_i))$. Thus,

$$\text{Sim}(\text{Context}_A.q, \text{Context}_B.q)=\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{n_i}\sum_{k=1}^{n_i}\text{CS}_{ik}.Q_A==\text{CS}_{ik}.Q_B\right),$$

where $n_i=\text{num}(C_A(\text{AS}_i))+\Delta\text{num}(C_B(\text{AS}_i))$.

$\text{CS}_{ik}.Q_A==\text{CS}_{ik}.Q_B$ represents whether the QoS information of a same candidate service CS_{ik} from Context_A to Context_B has changed. If no change occurs, the result is 1; if a change occurs, or CS_{ik} is withdrawn from Context_A to Context_B, or CS_{ik} is added from Context_A to Context_B, the result is 0. Thus, $\text{Sim}(\text{Context}_A.q, \text{Context}_B.q)$ reflects not only QoS information change from Context_A to Context_B, but also new services deployment or old services withdrawn from Context_A to Context_B.

3.3 Matching instances of execution plans

Instances of execution plans are matched by the execution plan matcher. The execution plan matcher searches existing solutions that may be reused for new unexecuted abstract services with similar properties. It is performed in two different situations:

1. Before a new composite service executes. When a new requested composite service comes for selection, this process searches the most recent successful execution plans for all the abstract services in this composite service.

2. When a composite service needs to be reselected. In this case, this process searches the most recent successful execution plans of the same composite service for the unexecuted abstract services.

The matched execution plans are used for the following context-aware historical execution plan selection step.

The properties considered in the searching process include the workflow of the composite service and the user context of the composite service, because these two properties highly affect global QoS driven service composition plans and can be easily computed.

It has been reported that an existing plan must have exactly the same workflow as the composite service to match a composite service that a user wants to execute, and whether two composite services are the same is decided by checking whether the abstract services are combined together using the same workflow pattern (Ye and Mounla, 2008). In this framework, if the matching process runs before a new composite service is executed, all the abstract services of the composite service need to be checked against the most recent successful execution plans (one of which is denoted as a plan). If the matching process runs when a composite service needs to be reselected in running, only the unexecuted abstract services of the composite service need to be checked with the execution plans of the same composite service.

The user context of the composite service affects the final QoS score of a given execution plan. In addition, it can be easily obtained from an execution plan. In this framework, the user context matching process runs between a new or running composite service (denoted by CS) and the most recent several successful execution plans after the workflow matching process. The user context of CS is denoted as W_{CSc}, W_{CSl}, W_{CSa}, W_{CSr}, and the user context of a plan is denoted as W_{Pc}, W_{Pl}, W_{Pa}, W_{Pr}. The user context matching process is to calculate the user context similarity between user context of CS (CS.u) and user context of plan (plan.u):

$$\mathrm{Sim}(\mathrm{CS}.u, \mathrm{plan}.u)=(|W_{CSc}-W_{Pc}|+|W_{CSl}-W_{Pl}| +|W_{CSa}-W_{Pa}|+|W_{CSr}-W_{Pr}|)/2.$$

The smaller the Sim(CS.u, plan.u), the higher the similarity between the user context of CS and the user context of the plan.

If the following condition holds, then the plan is matched with CS (denoted as Match(plan, CS)==true). A threshold $\varepsilon_{\mathrm{Sim}}$ is set such that the user context of the plan is regarded as similar to the user context of CS.

$$(\text{workflow of the plan matches workflow of CS}) \wedge(\mathrm{Sim}(\mathrm{CS}.u, \mathrm{plan}.u)\le\varepsilon_{\mathrm{Sim}}).$$

The instances of the current execution plan are stored and updated in the framework. Each instance consists of three elements: the workflow of the matched historical execution plans, the time when the execution of the plan finishes, and the transformed three types of context information of the execution plan.

3.4 Multi-population conflicts sorted repair genetic algorithm

Our MP-CS-repair GA adopts the rank-based selection mechanism, one-point crossover operator, one-point mutation operator, and information entropy theory based diversity maintaining mechanism (Cui and Lin, 2004) like other widely used repair GAs. We have improved the normal repair GA by a multi-population based (MP) initial individual generation mechanism and a conflicts sorted (CS) repair operator to make service re-plan effective and efficient, especially in the dynamic service execution environment with multiple and variable constraints.

3.4.1 Definitions

Definition 4 (Genome encoding) For each abstract service AS_i in a composite service ($1\le i\le n$, n is the number of abstract services in a composite service), all the available candidate services are obtained and marked with CS_{i1}, CS_{i2}, …, CS_{im}. The genome is encoded by an integer array with the number of items equal to the number of ASs. Each gene is the index of the chosen CS_{ij} for AS_i ranging from 1 to m_i (represented by j). m_i is the number of candidate services for the abstract service AS_i. The method is illustrated in Fig. 2. In our MP-CS-repair, the results of each generation are represented by a number of genomes. The genome corresponding to the maximal QoS score in the final generation is the result of service composition.

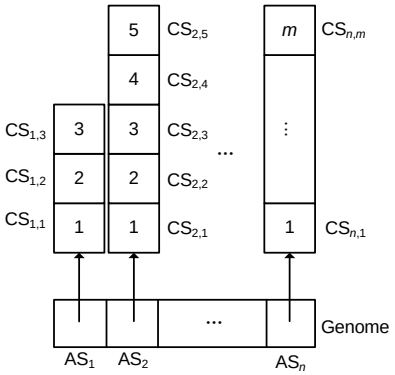

Fig. 2 Genome encoding in the MP-CS-repair GA

Definition 5 (State of candidate service) In the dynamic context sensitive environment, new candidate services may be deployed for each abstract service. Besides, the previous candidate services for each abstract service may be unavailable because some candidate services may disappear after an interval or the QoS of some candidate services may change to below a threshold. We use CS.state to represent whether a candidate service is available or not, CS.state∈{available, unavailable}. In MP-CS-repair GA, the state of CS in the genome (offspring) may be unavailable, so the genomes representing the previous service composition results may be unavailable and the previous service composition results may be reused not directly, but after a while, for a new workflow matched composite service.

Definition 6 (Fitness function) A fitness function is applied to evaluate the quality of the best individual in a generation according to the given optimization objective. In our approach, we use the global QoS (Eq. (3)) as the fitness function.

3.4.2 Multi-population based initial individual generation mechanism

When the matched instances for a composite service i is retrieved, the context-aware historical execution plan selector acquires final generation results (feasible service execution plans) for each matched instance from historical contexts and related execution plans in the framework. For service i, the most recent sets of final generations of historical service selection results are retrieved. Then the plan selector performs another context similarity analysis between the context of the current composite service and the context of each matched instance, and sorts the most recent sets of final generations of historical service selection results for composite i according to the context similarities. Denote WP_{ij} as the sorted results, where $1\leq j\leq num(WP_i)$, and $Sim(WP_{kj}.context, i.context)>Sim(WP_{lj}.context, i.context)$, where $1\leq k<l\leq num(WP_i)$. In addition, if $Sim(WP_{kj}.context, i.context)=1$, the execution plan in WP_{kj} with the highest fitness value will be re-used directly for the current composite service, and the MP-CS-repair GA terminates.

In this MP-CS-repair GA, not all of the initial individuals are newly generated. Some of them are selected from final generation results in similar context environments with a rank-based selection mechanism. To generate N initial individuals for workflow i, $(1/4)^j N$ individuals are selected from WP_{ij} with a rank-based selection mechanism, and the other individuals are newly generated. The reason for reusing previous generation results is that it can help the MP-CS-repair GA achieve a high fitness score in the initial generations. The randomness of the initial population is kept so that the final result of MP-CS-repair GA is not degraded.

3.4.3 Conflicts sorted repair operator

There are two reasons that result in the non-feasibility of the offspring in this GA: (1) the state of CS may be unavailable comparing the current CS and the previous ones in the generation process because the context condition may change; (2) the constraint context exists, and after the crossover and mutation process, the service selection results may violate the constraint conditions.

To enforce feasibility, we propose a repair operator, called the CS repair operator, to quickly repair the infeasible offspring. The process conducted by the CS repair operator is a heuristic approach to quickly replacing the selection results based on the current offspring, until all the constraint conditions are met. For a non-feasible offspring, the CS repairing process is shown below.

Algorithm 1 Conflicts sorted-based repairing

1. **foreach** CS_{ij} in the offspring such that CS_{ij}.state is unavailable **do**
2. randomly select CS_{ik}, such that CS_{ik}.state is available
3. **if** such CS_{ik} does not exist **then**
4. repair fails and exits
5. **end if**
6. **end for**
7. repair_times←0

8. **foreach** CS_{ij} whose CS_{ij}.state is feasible **do**
9. calculate the number of infeasible constraint context expressions that contain CS_{ij} (represented by CS_{ij}^c)
10. **if** $CS_{1j}^c = CS_{2j}^c = \ldots = CS_{\max j}^c = 0$ **then**
/*max: the number of abstract services*/
11. repair succeeds and exits
12. **end if**
13. **if** repair_times>repair_max_times **then**
14. repair fails and exits
15. **end if**
16. sort the infeasible constraint context expressions according to the number of related ASs in each expression (denoted as E^{AS})
17. randomly choose a context expression E with its E^{AS} being the maximum
18. randomly choose an AS_i related to E
19. randomly select a $CS_{ik'}$ for AS_i such that $CS_{ik'} \neq CS_{ij'}$ and $CS_{ik'}^c$ is the smallest
20. **if** $CS_{ik'}$ does not exist **then**
21. repair fails and exits
22. **end if**
23. replace $CS_{ij'}$ by $CS_{ik'}$
24. repair_times←repair_times+1
25. **end for**

If the state of a CS in the offspring is unavailable, an initial repairing process (lines 1–6) is required to make the state of all the CS available. There is a heuristic repairing process in lines 8–25, where the constraint expressions are sorted and the context constraints influencing the most number of candidate services for selection are considered first in each repairing process. This context expression sorted repairing process can increase the success rate while keeping the diversity of individuals during the repairing process.

4 Experiments

4.1 Prototype implementation

We implemented a prototype for this proposed framework in Java using Eclipse in Windows XP platform on an Intel Core 2 Duo 2.4 GHz machine with 2 GB memory. The user and service contexts were stored in XML files while policy and device contexts were stored using JBoss Rule structure files. All the files storing different kinds of QoS related context were generated periodically with different cycles to simulate a dynamic context environment. The input workflows for composition were the BPEL abstract specifications. The four different components in the prototype, context manager, service selection/re-selection trigger, QoS-driven context-based service selector/re-selector, and composite service executor, can run concurrently and coordinate with each other using Java signals. The execution plan matcher module was implemented using the SHA1 hash function with BPEL files representing workflows and we used MYSQL as the database.

4.2 Performance analysis of the MP-CS-repair GA based service selector/re-selector

We tested the performance of the service selector/re-selector which performs the MP-CS-repair GA in this prototype. The workflow for composition (Fig. 3) was described by BPEL files, which contained 10 distinct abstract services and involved four types of service composition structure: sequential, parallel, branch, and loop. Different service composition problems with different numbers of abstract services can be constructed: to construct a service composition problem with 20 abstract services, we concatenated two blocks of this workflow, to construct a service composition problem with 30 abstract services, we concatenated three blocks of this workflow, and so on. In this way, we constructed 10 service composition problems whose abstract service numbers ranged from 10 to 100 with an increment of 10. Each path of the same branch service composition structure in this workflow had a uniform execution probability and the number of iterations of the loop structure in the workflow was five.

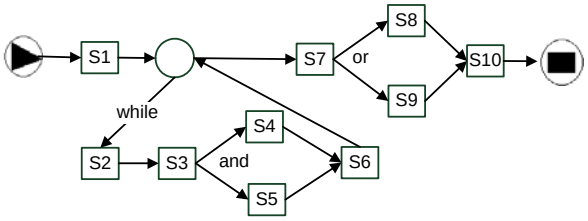

Fig. 3 The workflow for composition in the experiments described by BPEL files, containing 10 distinct abstract services and involving four types of service composition structure: sequential, parallel, branch, and loop

The QoS values for candidate services stored in the XML files describing service context were generated randomly. The ranges of the QoS values for cost, latency, availability, and reliability were [1, 10], [1, 10], [0.9, 1], and [0.9, 1], respectively, and the four

weights of these criteria were all fixed to 0.25. The XML files containing service contexts and JBoss Rule structures containing policy and device contexts were generated randomly. In addition, the contexts generated did not contradict with each other, and service selection results still existed under all the contexts.

4.2.1 For a new service composition problem

In this experiment the performance of the MP-CS-repair GA was analyzed for solving new service composition problems in a composition situation. The composition situations simulated in XML files, JBoss Rule structures, and BPEL files contained abstract services in one or more blocks of the workflow as shown in Fig. 3, the collected QoS information of candidate services, and transformed context information in the form of constraint and unconstraint context expressions. The proportion of the device constraint, exclusive constraint, single constraint, combined QoS change, and user satisfaction enhancement context expression number was 1:10:2:1:1, and there were four combined QoS constraint expressions.

For a new service composition problem, no matched instance was acquired by the execution plan matcher and we tested only the runtime from the time when the composition situations were transformed and stored in the memory to the time when the composition results were generated. The parameters of MP-CS-repair GA were set as follows: initial population size, 500; population size, 50; crossover probability, 0.9; mutation probability, 0.08; repair_max_times, 50; maximum number of generations, 1000. Each result was averaged over 10 test cases. Figs. 4–6 show the experiment results, exploring the scalability, extensibility, and efficiency of the MP-CS-repair GA based service selector/re-selector. Though fluctuating, the computation time increased almost linearly with the increase of the numbers of abstract services, candidate services for each abstract service, and constraint expressions.

4.2.2 For a service re-plan problem

To study the performance of the re-selector in solving a service re-plan problem, we considered a simple case in our prototype. Suppose after the execution of the first task S1 in the workflow, the candidate service of S2 and its backup service both became unavailable, while all the other candidate services for each task were available and all context

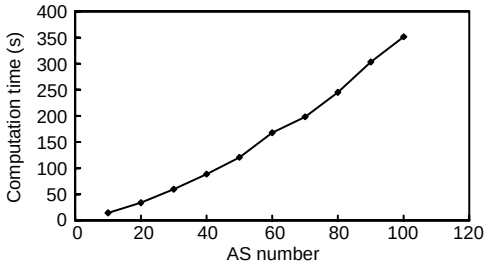

Fig. 4 The impact of number of abstract services (ASs) on computation time (candidate service number for each AS: 50; constraint expression number: 4)

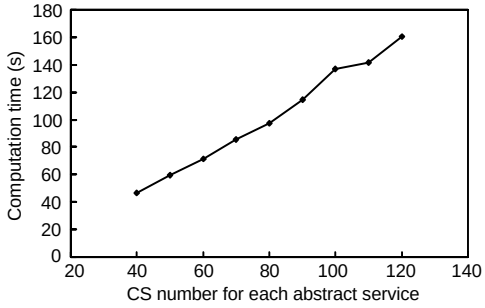

Fig. 5 The impact of the number of candidate services (CSs) on the computation time (abstract service number: 20; constraint expression number: 420)

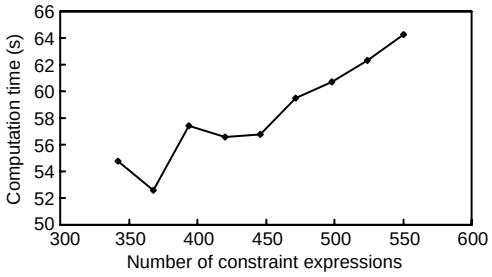

Fig. 6 The impact of the number of constraint expressions on the computation time (abstract service number: 20; candidate service number for each abstract service: 50)

information and QoS of candidate services did not change from the time when the whole composite service was executed. A service re-plan process was triggered for S2 and all the other unexecuted tasks by the MP-CS-repair GA. The most recent sets of final generations of historical service selection results could be re-used in the initial individual generation process of the MP-CS-repair GA.

According to our current investigation, only the GA with a repair operator can be suitable for solving this optimizing problem in a dynamic context environment with dynamic and complicated conditions and constraints, and all the conditions and constraints

may change when the algorithm is running. Thus, comparative experiments were conducted using the minimal-conflict hill-climbing (MCHC) repair GA (Ai and Tang, 2008a), an improved penalty-based GA (Ai and Tang, 2008b) (with a repair operator in MCHC-repair GA in its initial individuals), and our MP-CS-repair GA on the re-selector of varying numbers of unexecuted abstract services, candidate services, and constraint expressions. The mechanism for generating a composition environment and the parameter setting in MP-CS-repair GA were the same as in the experiment above. Also, each result was averaged over 10 tests.

As depicted in Tables 2 and 3, our MP-CS-repair GA outperformed the MCHC-repair GA and the improved penalty-based GA for a service re-plan problem. As the abstract service number, candidate service number for each abstract service, and constraint context expression number increased, our MP-CS-repair GA saved more computation time for arriving at or exceeding 99% of the global QoS of the MCHC-repair GA and the improved penalty-based GA.

Fig. 7 plots average fitness evolution values across GA generation for one test case in Table 2 or Table 3 where the abstract service number was 19, the candidate service number was 50, and the device number was 10. The detailed service selection results, backup results, and re-plan results at runtime are shown in Table 4. The average values across GA generation were obtained by running each GA 20 times. The relationship between generation and fitness values for the three GAs was analyzed. Each GA terminated at the arrival of 1000 generations, and its final average global QoS for the 20 tests was recorded.

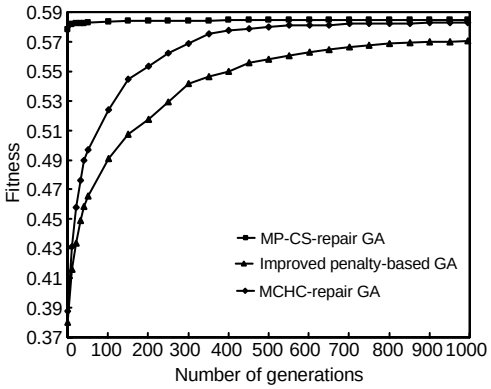

Fig. 7 Relationship between the number of generations and fitness for the three genetic algorithms, with 19 abstract services, 50 candidate services, and 10 devices

As depicted in Fig. 7, the fitness of MP-CS-repair GA was very close to the optimum after its first several generations. This reveals the advantage of re-using final generations of service composition results in a similar context environment in initial populations in MP-CS-repair GA. Thus, this algorithm is very suitable for a global optimizing service re-plan problem to decrease the extra delay caused in re-plan if the current running service fails while the whole service composition environment has not undergone a radical change.

Table 2 Comparison between MCHC-repair GA and MP-CS-repair GA of varying numbers of abstract services (ASs), candidate services (CSs), and constraint expressions (CEs) for a service re-plan problem

Device number	AS number	CS number	CE number	t_a (ms)	t_b (ms)	Avg. (t_b/t_a)	Avg. $F(x_b)/F(x_a)$
10	9	50	277	14469	7637	0.525523	0.993285
10	19	50	420	65255	31228	0.495654	0.990492
10	29	50	563	148147	65922	0.460861	0.997858
10	39	50	706	372488	171070	0.449075	1.010427
10	19	40	420	47542	26580	0.629019	0.990593
10	19	60	420	71859	29545	0.442504	0.998787
10	19	80	420	119839	40605	0.341886	0.992315
10	19	100	420	179411	74405	0.403844	0.993719
10	19	120	420	198124	68783	0.358876	1.001026
4	19	50	342	62300	34130	0.502556	0.996014
8	19	50	394	56039	29577	0.547375	0.992612
12	19	50	446	51855	20411	0.430488	0.996800
16	19	50	498	56042	23986	0.441419	0.997975
20	19	50	550	54609	25867	0.469852	0.997887

t_a: runtime for MCHC-repair GA; t_b: runtime for the proposed MP-CS-repair GA. F: fitness function

Table 3 Comparison between the improved penalty-based GA and MP-CS-repair GA of varying numbers of abstract services (ASs), candidate services (CSs), and constraint expressions (CEs) for a service re-plan problem

Device number	AS number	CS number	CE number	t_c (ms)	t_d (ms)	Avg. (t_d/t_c)	Avg. $F(x_d)/F(x_c)$
10	9	50	277	12500	9402	0.768875	0.989977
10	19	50	420	58623	29617	0.533040	0.999273
10	29	50	563	133561	58750	0.449512	1.037914
10	39	50	706	273530	124744	0.445673	1.047992
10	19	40	420	42339	21602	0.529606	1.011781
10	19	60	420	60897	29483	0.485872	1.010571
10	19	80	420	107961	51806	0.498351	0.999070
10	19	100	420	143464	61189	0.423902	1.008572
10	19	120	420	168664	71555	0.427105	1.007516
4	19	50	342	58755	33136	0.575123	1.005631
8	19	50	394	57010	31797	0.586348	0.999949
12	19	50	446	51048	21135	0.429483	1.004630
16	19	50	498	50286	27534	0.540305	1.019236
20	19	50	550	59494	28766	0.491610	1.006521

t_c: runtime for improved penalty-based GA; t_d: runtime for the proposed MP-CS-repair GA. F: fitness function

Table 4 Re-plan of the composition process at runtime

Task	Index of the selected candidate service for each abstract service				
	Initial process	Initial backup process	MCHC-repair GA*	MP-CS-repair GA*	Improved penalty-based GA*
S1	25	5			
S2	29	44	9	9	44
S3	33	23	33	33	33
S4	13	33	13	13	13
S5	46	30	27	46	27
S6	35	45	45	35	45
S7	22	41	22	10	22
S8	35	8	35	35	35
S9	32	44	32	32	32
S10	48	40	48	48	48
S11	46	22	46	46	46
S12	32	24	32	32	32
S13	18	23	23	18	23
S14	28	26	26	26	26
S15	2	46	46	2	46
S16	45	1	45	45	45
S17	11	40	40	11	40
S18	39	43	39	39	39
S19	12	6	9	12	9
S20	46	49	49	21	23
Global QoS	0.586520	0.518486	0.582925	0.584536	0.572220

* One re-planning result. Index of the selected candidate service for the executed partial flow S1 is 25

4.3 Performance analysis of the context-aware service re-selection mechanism

4.3.1 Effectiveness and extensibility

The four different components in this prototype (Fig. 1) were running concurrently to realize this mechanism. Two concatenated blocks of workflow (the number of interaction times of loop structure in the workflows was modified to one) in Fig. 3 after the initial composition were running by the composite service executor, while their related context was changing by generating and updating XML files and JBoss Rule structures automatically. The context expressions representing the inner device resource constraints, the resource constraints between devices, the single constraints, the combined QoS change, and the user satisfaction enhancements were all changing concurrently every T_1 per context expression, and T_1 satisfied the Gaussian distribution N(C1, C1/2) (represented by C1 context change cycles). Every T_2, a new candidate service would be added for a randomly selected AS or an existing candidate service would be removed for a randomly selected AS, T_2~N(C2, C2/2) (represented by C2 cycles). Every T_3, a randomly selected candidate service for a randomly selected AS would change its QoS parameters, T_3~N(C3, C3/2) (represented by C3 cycles). Other constraint contexts including exclusive constraints and combined QoS constraints were not considered in this experiment.

Three service composition mechanisms were compared in the same dynamic environment, including our context-aware service re-selection mechanism ('re-selection' for short), a normal service composition mechanism without the backed up replacement service for each task ('normal' for short), and a service backup mechanism with the backed up replacement service for each task ('backup' for short). The parameters for MP-CS-repair GA were the same as in Section 4.2.1. The range of the values for latency for each CS was [10, 100] s. The maximum number of generations of MP-CS-repair GA was set to 500, and C1:C2:C3=30:2:1. The service re-selection triggering cycle in the proposed mechanism was set to 5 s. User context did not change when the composite service was running: W_c and W_q were both set to 0.5. To measure global QoS, $V_{\max}$ and $V_{\min}$ were obtained before the composite service was running and remained unchanged. Each result was averaged over five different test cases. Figs. 8–11 explore the effectiveness and extensibility of the re-selection mechanism.

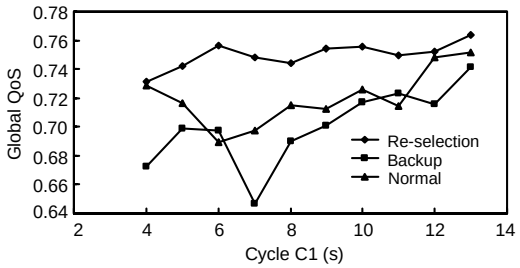

Fig. 8 Global QoS for the executed composite service (CS) as context change cycle varies (CS number for each abstract service: 50; context change threshold: 0.8)

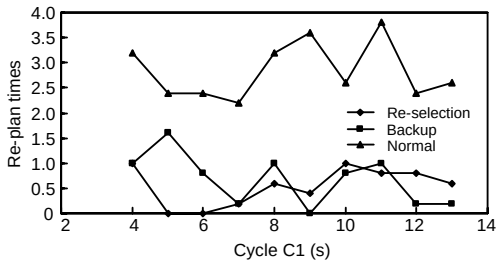

Fig. 9 Re-plan times for the running composite service (CS) as context change cycle varies (CS number for each abstract service: 50; context change threshold: 0.8)

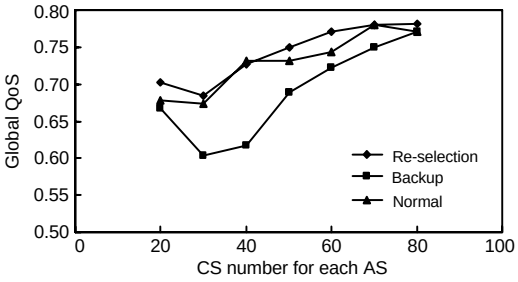

Fig. 10 Global QoS for the executed composite service (CS) as the CS number for each abstract service (AS) varies (context change cycle C1: 8 s; context change threshold: 0.8)

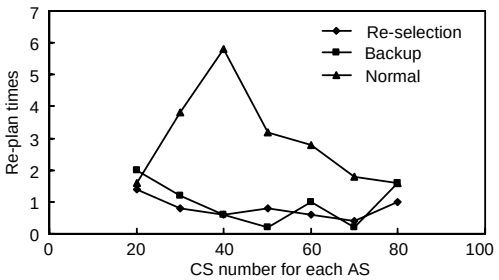

Fig. 11 Re-plan times for the running composite service (CS) as the CS number for each abstract service (AS) varies (context change cycle C1: 8 s; context change threshold: 0.8)

As the context change cycles varied, the global QoS for the running composite service (the time spent in re-plan was not included in global QoS) in our proposed mechanism exceeded those in the normal and backup mechanism cases. The number of re-plan times for the running composite service in the re-selection mechanism case remained as low as that in the backup mechanism case. In addition, since the MP-CS-repair GA outperforms other GAs in reducing the time for convergence in the service re-plan (Section 4.2.2), the overall extra delay caused by the re-plan can be reduced. In the experiment, the QoS information for each executed CS was determined when the CS finished. In reality, the benefit of the proposed mechanism will be more obvious as some QoS attributes in the global QoS are determined when each selected CS starts to be executed, without a QoS degradation process when the CS is running. In these situations, the context-aware service re-selection mechanism will have more effect in increasing global QoS.

4.3.2 Re-selection threshold and adaptability

This experiment was aimed to explore the impact of the re-selection threshold (context change threshold in Fig. 1) on the global QoS for executed CS for the proposed mechanism. Four components in this prototype were running in the same environment as in Section 4.3.1, except that the ratio of context change cycles C1:C2:C3 was modified to 10:2:1. Each result was averaged over five test cases in the same service composition and context change environment. As depicted in Figs. 12 and 13, when the re-selection threshold was very low, global QoS for executed CS varied and was not affected. The global QoS increased as the threshold increased, but required increasing computation cost in re-selection. The threshold cannot be too high because it resembles the approach of re-selecting services every time a change occurs. It is not feasible due to the high time complexity of the re-selection (Canfora *et al*., 2005b), which interrupts the execution of the composite service and affects its performance. The threshold can be adjusted according to different users' requirements of global QoS of the executed composite service for reducing computation cost in re-selection and making the mechanism flexible.

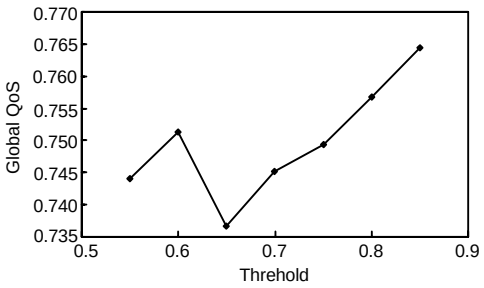

Fig. 12 The impact of the re-selection threshold on the global QoS for the executed composite service (composite service number for each abstract service: 50; context change cycle C1: 4 s)

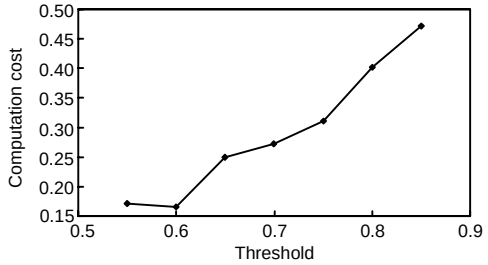

Fig. 13 Relationship between the computation cost in re-selection (represented as the ratio of the computation time in re-selection to the total computation time) and the re-selection threshold (composite service number for each abstract service: 50; context change cycle C1: 4 s)

5 Related work

In the research fields of QoS-aware service composition considering context information, it is reported that a global QoS-aware service selection problem was modeled as the multiple-choice knapsack problem (MCKP) and multi-dimensional multiple-choice knapsack problem (MMKP) to maximize the user-defined utility function value while meeting the end-to-end delay constraint (a combined QoS constraint in policy context) (Yu and Lin, 2005c; Yu *et al*., 2007). A method based on branch-and-branch was proposed to obtain the best solution of the MMKP problem. However, the MCKP and MMKP based models cannot handle the combined QoS change of policy context issues, in which the QoS of one candidate service may depend on the service selection results of other abstract services in the workflow. The global QoS-aware service selection problem was modeled by Zeng *et al*. (2004) with multiple global QoS constraints, and a linear inter-programming

approach was proposed for solving the optimization problem. The model was also extended by a mixed integer linear programming problem (Ardagna and Pernici, 2007). However, their proposed inter programming approach still cannot handle the combined QoS change of policy context issues. Ye *et al*. (2008) proposed a global QoS constrained service-correlation aware QoS model to characterize the QoS dependence of an individual service on the other related services. They proposed an improved inter-programming based selection approach for optimal solution, and a heuristic based selection approach for sub-optimal solution to the global QoS optimization problem. Their service-correlation aware QoS model can handle the combined QoS change of policy context issues, and the proposed heuristic based selection approach can solve a global QoS optimization service selection problem with multiple context constraints. Some researchers focused on the resource or device context constraints imposed on the QoS-aware service composition problem. Tsesmetzis *et al*. (2007) studied the problem of service provision when a service provider receives concurrent requests of numerous customers for Web services demonstrating different bandwidth and price properties, and introduced the selective MCKP to identify the services that should be delivered to the customers, in order to maximize the provider's profit, subject to maximum bandwidth constraints on the server side. The QoS-awareness was limited to the consideration of two QoS parameters: service bandwidth and cost.

Zhou *et al*. (2008) considered more context information influencing the result of service composition, where the conventional context-free QoS selection model was improved by a policy-based context sensitive QoS model, considering context sensitive constraints and the changeability of QoS values, to effectively calculate QoS values and select Web services. Their proposed business rule categorizations for context representation were re-used in policy context in this study. Service context, device context, and user context in a context sensitive service composition problem (Tang *et al*., 2008) were considered based on a context negotiation supporting pervasive computing environment, and a local optimization algorithm was proposed for each task in the composite service to improve resource utilization.

In a context sensitive QoS-aware service composition environment, as context information becomes more complicated and the context constraint number increases, GA outperforms the integer programming based methods in effectiveness, efficiency, and adaptability in a dynamic environment with multiple and variable constraints. In addition, GA shows a better performance and scalability than linear integer programming with increasing numbers of candidate Web services and tasks (Canfora *et al*., 2005a), and compared with integer programming methods (Zeng *et al*., 2004; Berbner *et al*., 2006; Tsesmetzis *et al*., 2007; Yu *et al*., 2007), the optimization objective function of GA need not be linear. There were many recent studies (Zhang *et al*., 2006; Gao *et al*., 2007; Liu *et al*., 2007; Ai and Tang, 2008a; 2008b; Canfora *et al*., 2008; Gong *et al*., 2008) which improved a traditional GA to solve QoS-aware service composition in service planning or re-planning to optimize global QoS for the composite service. The improved GAs showed effectiveness, efficiency, scalability, or extensibility in supporting global QoS-aware service composition with context constraints.

The researches above mainly regarded context information as static constraints or conditions for service composition before composite service running or re-execution. In a pervasive computing environment, context information for service composition is not static, which results in the global QoS of running service composition plan degrading from the optimum. The easiest solution to this problem is to re-select the services once a context change occurs, which is time consuming, and the execution of the composite service will be interrupted frequently (Dai *et al*., 2009). Canfora *et al*. (2005b) was another study of re-selection in the execution of the composite service, where the re-selection is triggered as soon as the actual QoS deviates from the initial estimates. The execution of the composite service will not continue until the re-selection finishes; however, this approach can be used only for runtime-unaware applications (Dai *et al*., 2009). Other researchers (Reiff-Marganiec *et al*., 2007; Yu *et al*., 2008) made service selection decisions activity by activity based on the existing local and global composition context for each abstract service in the composition service. This method needs to invoke the evaluation framework for each abstract service in a composite service, which will cause an extra delay. Thus, it is still suitable only for runtime-unaware applications. To minimize the extra

delay caused by re-plan, the solution of backing up a replacement for each task was proposed (Yu and Lin, 2005b; Girish *et al*., 2006); when one task in the composite server insures a failure, the composite service can easily switch to a replacement, avoiding an extra delay. Recent researchers (Dai *et al*., 2009) improved the availability of replacement for composite services by performance prediction, which enables the composite service to heal by itself from a failure as quickly as possible and minimizes the number of re-plans. The performance prediction was focused on the transmission speed in supporting the modeling reliability of services for better QoS and higher reliability of replacement for composite services.

6 Conclusions and future work

QoS-aware service composition is aimed to maximize global QoS for all the tasks for execution in composition. In a context sensitive service execution environment, the context constraints and conditions affecting the service execution plan are not static, which results in the current service execution plan failing or its QoS degrading from the current optimum. This paper deals with a runtime self-optimizing service composition approach, including its framework and prototype implementation in a context sensitive environment. The context-aware service re-selection mechanism is included in this framework based on context similarity analysis to improve global QoS for all the tasks for execution in composition, and an MP-CS-repair genetic algorithm is incorporated into the framework to reduce extra delay in computation when a service re-plan occurs. The effectiveness, scalability, extensibility, and efficiency are validated by comparative experiments on a prototype for this framework, and this context-aware service re-selection mechanism also shows adaptability to context change frequencies and user requirements. Currently, the QoS and context information of each candidate service is deterministic. In the future, we will consider the uncertainty of QoS and context information in QoS-aware service composition as well as a context conflicts solving mechanism.

References

Ai, L., Tang, M., 2008a. QoS-Based Web Service Composition Accommodating Inter-service Dependencies Using Minimal-Conflict Hill-Climbing Repair Genetic Algorithm. IEEE 4th Int. Conf. on eScience, p.119-126. [doi:10.1109/eScience.2008.110]

Ai, L., Tang, M., 2008b. A Penalty-Based Genetic Algorithm for QoS-Aware Web Service Composition with Inter-service Dependencies and Conflicts. Int. Conf. on Computational Intelligence for Modelling, Control and Automation; Intelligent Agents, Web Technologies and Internet Commerce, and Innovation in Software Engineering, p.738-743. [doi:10.1109/CIMCA.2008.104]

Ardagna, D., Pernici, B., 2007. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, **33**(6):369-384. [doi:10.1109/TSE.2007.1011]

Berbner, R., Spahn, M., Nicolas, R., Oliver, H., Steinmetz, R., 2006. Heuristics for QoS-Aware Web Service Composition. Int. Conf. on Web Services, p.72-82. [doi:10.1109/ICWS.2006.69]

Canfora, G., Esposito, R., di Penta, M., Villani, M.L., 2004. A Lightweight Approach for QoS-Aware Service Composition. Proc. 2nd Int. Conf. on Service Oriented Computing, p.1-10.

Canfora, G., di Penta, M., Esposito, R., Villani, M.L., 2005a. An Approach for QoS-Aware Service Composition Based on Genetic Algorithms. Genetic and Evolutionary Computation Conf., p.1069-1071. [doi:10.1145/1068009.1068189]

Canfora, G., di Penta, M., Esposito, R., Villani, M.L., 2005b. QoS-Aware Re-planning of Composite Web Services. Proc. Int. Conf. on Web Services, p.121-129. [doi:10.1109/ICWS.2005.96]

Canfora, G., di Penta, M., Esposito, R., Villani, M.L., 2008. A framework for QoS-aware binding and re-binding of composite Web services. *J. Syst. Softw.*, **81**(10):1754-1769. [doi:10.1016/j.jss.2007.12.792]

Cui, X., Lin, C., 2004. Multicast QoS routing optimization based multi-objective genetic algorithm. *J. Comput. Res. Dev.*, **41**(7):1144-1150 (in Chinese).

Dai, Y., Yang, L., Zhang, B., 2009. QoS-driven self-healing Web service composition based on performance prediction. *J. Comput. Sci. Technol.*, **24**(2):250-261. [doi:10.1007/s11390-009-9221-8]

Gao, C., Cai, M., Chen, H., 2007. QoS-Driven Global Optimization of Services Selection Supporting Services Flow Re-planning. APWeb/WAIM Workshops, p.516-521. [doi:10.1007/978-3-540-72909-9_56]

Girish, C., Koustuv, D., Arun, K., Sumit, M., Biplav, S., 2006. Adaptation in Web Service Composition and Execution. Proc. Int. Conf. on Web Services, p.549-557.

Gong, X., Zhu, Q., Wu, C., Lin, L., 2008. Web services composition supporting global optimal and dynamic re-planning of QoS. *Comput. Integr. Manuf. Syst.*, **14**(10):2068-2075 (in Chinese).

Hu, J., Tang, C., Duan, L., Zuo, J., Peng, J., Yuan, C., 2007. The strategy for diversifying initial population of gene expression programming. *Chin. J. Comput.*, **30**(2):305-310 (in Chinese).

Keidl, M., Kemper, A., 2004. Towards Context-Aware Adaptable Web Services. Proc. 13th Int. World Wide Web Conf. on Alternate Track Papers and Posters, p.55-65. [doi:10.1145/1013367.1013378]

Liu, S., Liu, Y., Zhang, F., Tang, G., Jing, N., 2007. A dynamic Web services selection algorithm with QoS global optimal in Web services composition. *J. Softw.*, **18**(3):646-656 (in Chinese). [doi:10.1360/jos180646]

Reiff-Marganiec, S., Yu, H.Q., Tilly, M., 2007. Service Selection Based on Non-functional Properties. Int. Conf. on Service-Oriented Computing, p.128-138. [doi:10.1007/978-3-540-93851-4_13]

Renders, J.M., Flasse, S.P., 1996. Hybrid methods using genetic algorithms for global optimization. *IEEE Trans. Syst. Man Cybern. Part B*, **26**(2):243-258. [doi:10.1109/3477.485836]

Tang, L., Huai, X., Li, M., 2008. An approach to dynamic service composition based on context negotiation. *J. Comput. Res. Dev.*, **45**(11):1902-1910 (in Chinese).

Tsesmetzis, D., Roussaki, I., Efstathios, S., 2007. Modeling and simulation of QoS-aware Web service selection for provider profit maximization. *Simulation*, **83**(1):93-106. [doi:10.1177/0037549707079229]

Ye, S., Wei, J., Li, L., Huang, T., 2008. Service-correlation aware service selection for composite service. *Chin. J. Comput.*, **31**(8):1383-1397 (in Chinese). [doi:10.3724/SP.J.1016.2008.01383]

Ye, X., Mounla, R., 2008. A Hybrid Approach to QoS-Aware Service Composition. IEEE Int. Conf. on Web Services, p.62-69. [doi:10.1109/ICWS.2008.29]

Yu, H.Q., Reiff-Marganiec, S., Tilly, M., 2008. Composition Context for Web Services Selection. IEEE Int. Conf. on Web Services, p.785-786. [doi:10.1109/ICWS.2008.98]

Yu, T., Lin, K.J., 2005a. Service Selection Algorithms for Composing Complex Services with Multiple QoS Constraints. Proc. Int. Conf. on Service-Oriented Computing, p.130-143. [doi:10.1007/11596141_11]

Yu, T., Lin, K.J., 2005b. Adaptive Algorithms for Finding Replacement Services in Autonomic Distributed Business Processes. Proc. Int. Symp. on Autonomous Decentralized Systems, p.427-434. [doi:10.1109/ISADS.2005.1452105]

Yu, T., Lin, K.J., 2005c. Service selection algorithms for Web services with end-to-end QoS constraints. *Inform. Syst. e-Business Manag.*, **3**(2):103-126. [doi:10.1007/s10257-005-0052-z]

Yu, T., Zhang, Y., Lin, K.J., 2007. Efficient algorithms for Web services selection with end-to-end QoS constraints. *ACM Trans. Web*, **1**(1):6-32. [doi:10.1145/1232722.1232728]

Zeng, L.Z., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H., 2004. QoS-aware middleware for Web services composition. *IEEE Trans. Softw. Eng.*, **30**(5):311-327. [doi:10.1109/TSE.2004.11]

Zhang, C., Su, S., Chen, J., 2006. Genetic algorithm on Web services selection supporting QoS. *Chin. J. Comput.*, **29**(7):1029-1037 (in Chinese).

Zhou, T., Zheng, X., Song, W.W., Du, X., Chen, D., 2008. Policy-Based Web Service Selection in Context Sensitive Environment. IEEE Congress on Services: Part I, p.255-260. [doi:10.1109/SERVICES-1.2008.30]