QoS-Based Service Provision Schemes and Plan Durability in Service Composition

Koramit Pichanaharee and Twittie Senivongse

Department of Computer Engineering, Faculty of Engineering, Chulalongkorn University
Phyathai Road, Pathumwan, Bangkok 10330 Thailand
koramit.p@gmail.com, twittie.s@chula.ac.th

Abstract. In service composition, quality of service is a major criterion for selecting services to collaborate in a process flow to satisfy a certain quality goal. This paper presents an approach for service composition which considers QoS-based service provision schemes and variability of the QoS when planning. The QoS of a service can be stated in terms of complex service provision schemes, e.g. its service cost is offered at different rates for different classes of processing time, or its partnership with another service gives a special class of QoS when they operate in the same plan. We also address that it is desirable for service planning to result in a plan that is durable and reusable since change in the plan, e.g. by deviation of the actual QoS, would incur overheads. Our planning approach takes into account these dynamic situations and is demonstrated by using the Estimation of Distribution Algorithm.

Keywords: Service Composition, QoS, Estimation of Distribution Algorithm.

1 Introduction

Service composition is a process that selects software units, called services, and composes them into a workflow that represents a business process [1]. The workflow can be viewed as a composite service since it provides an aggregated function and can be used further in composition of other services or business processes. A service composition problem can be considered a planning problem. That is, given a flow of abstract services (AS) for a particular business domain as a goal, composition will create a plan that satisfies such a goal by assigning a service instance (SI) in place of each abstract service. A flow for a travel planner, for example, may consist of three abstract services, i.e. tourist information, transportation, and accommodation services. A service instance will be selected for each abstract service to make a concrete plan.

Quality of service (QoS) has been considered widely in the composition problem. Service instances that collectively give the optimal quality or meet the quality defined by the user will be the solution to planning. Several publications [2-5] give slightly different QoS definitions but the most common QoS attributes include cost, time, availability, reliability, reputation, and security of services. A number of research efforts have proposed ways to compose services based on QoS attributes by using

R. Meier and S. Terzis (Eds.): DAIS 2008, LNCS 5053, pp. 58–71, 2008.
© IFIP International Federation for Information Processing 2008

several optimisation methods and techniques, e.g. integer programming, linear programming, genetic algorithm. We are interested in using QoS attributes to determine the solution plan, but also discuss the following issues:

1. The QoS of service instances may vary when the instances are used in different operational environments, and therefore different users may have different experiences when using the same service instance. To have an accurate view of the actual QoS, each user will maintain his/her own experiences in using particular service instances, i.e. how much the actual QoS deviates from what was published by service providers. Such information should be fed into future planning which involves those service instances.
2. In certain cases, an organisation that formulates a composition plan to represent a business process would expect the plan to last and can be reused at least for some time. A travel planning organisation, for example, would use the plan which consists of particular instances of tourist information, transportation, and accommodation services to arrange trips for its customers. If the actual QoS of these instances is quite stable compared to what was published at planning time, the organisation can reuse the plan in serving its customers. Deviation of the QoS of each service instance can affect the overall QoS of the flow and hence will call for a new plan to be composed. Such replanning incurs overheads and is not desirable if it happens frequently. We address this issue as plan durability.
3. Instead of publishing service quality in terms of individual QoS attributes, service providers can state the QoS in terms of complex service provision schemes to realise various classes of service provision. For example, a service instance may offer different classes of service time which vary by the cost charged to the user, or the availability of the service instance may vary by the time of use. A service provider may also work in partnership with another provider to provide a special class of service, e.g. they give a discount in cost or offer less service time if their instances operate together in a plan. Partnership nevertheless can affect plan durability. When two service instances are coupled, a QoS deviation at one instance could affect its partner and lead to change at both instances; the plan becomes less durable as a single deviation may incur change to a large extent of the plan.

In this paper, we propose an extended QoS model and a planning approach that will result in composition plans that address the three situations above. QoS-based service provision schemes will be taken into account when service instances are selected for the plans. Our QoS model captures common quality attributes, i.e. cost, time, reliability, and availability, and enhances with durability quality via self-rating and partnership-coupling metrics. Self-rating refers to the rating an individual user gives to a particular service instance based on his/her own experience in its QoS. Partnership coupling refers to the degree of coupling between service instances which is present in the plan via partnership schemes. We see that considering the durability issue at planning time will result in a plan that lasts longer and thus help reduce the chance of frequent replanning. We use the Estimation of Distribution Algorithm (EDA) [6], which is a technique of the Genetic Algorithm (GA) [7], as the planning algorithm. To search for a planning solution, the EDA generates population by a probabilistic

model which is derived from the knowledge obtained from the past generations of population. We are interested in this characteristic of the EDA because such knowledge should facilitate the creation of a new population during a solution search process, and should additionally contribute to replanning when a new plan is needed. A simulation of service instances QoS and EDA-based planning will be conducted.

Our approach can be used for both offline and runtime planning. A concrete plan is composed with regard to a given abstract flow, service provision schemes, and the durability issue. The plan should last until there is a requirement for a new plan, e.g. when there are new service instances or new updates on the QoS of existing instances. If runtime monitoring of the flow is supported, planning can also be triggered when there are service outages or serious deviations of the QOS. This work assumes that all service instances that are bound to a plan are compatible in terms of interface signatures and semantics. Service providers will publish the QoS of service instances in a public registry, e.g. UDDI for Web services, or provide other means for users to have access to QoS attributes.

The paper is organised as follows. Section 2 presents related work and Section 3 discusses our approach to QoS-based service provision schemes and variability of the QoS. We present the extended QoS model in Section 4 and describe how EDA is used in planning in Section 5. Simulation results from running the EDA are shown in Section 6 and the paper concludes with a discussion and future work in Section 7.

2 Related Work

Research works in QoS-based service planning tackle this problem by using different optimisation techniques to find composition plans based on slightly different QoS models. One of the major efforts in this area is the work in [2] which proposes a QoS model that captures execution price, execution duration, reputation, successful execution rate, and availability. By using integer programming, its QoS-aware composition maximises the QoS of composite services while taking into account the constraints of the users. Its supporting execution environment also considers runtime changes in the QoS of the service instances. The QoS model in [8] is used as a fitness function for composing a plan by GA. The QoS attributes include time, cost, reliability, and availability, and the plan will be penalised if it violates user QoS constraints. In [9], time, cost, and reliability are of concern in the QoS model and a distance function-based multi-objective evolutionary algorithm is used to find an optimised composition. A QoS reference vector is proposed in [10] to model price, time, reliability, trust (i.e. subjective rating), and security. The work evaluates service quality against cost of service selection by comparing a global exhaustive search and the integer programming approach. The work in [11] introduces a model-driven methodology for building QoS-optimised composite services and uses UML profile for QoS to model QoS requirements. The overall QoS of a plan is determined based on the multiple criteria decision making approach and patterns of control flow. Price, execution time, user rating, and encryption level are the QoS attributes of concern.

The paper [12] proposes a broker that supports planning and execution of any composite services with multiple QoS classes. Since a particular plan can be reused and executed repetitively as a flow, the QoS can be guaranteed on a per-flow rather

than a per-request basis, and different QoS levels can be negotiated with respect to the volume of execution requests. Time, cost, and availability are included in the QoS model, and linear programming is used as the planning algorithm. The work in [13] presents a semantics-based planning approach in which data semantics, functional semantics, QoS (i.e. time, cost, reliability, availability, domain-specific QoS metrics), and constraints of service instances are considered. Ontology-based service dependencies such as business/technological constraints and partnership between services are addressed, and integer linear programming is used as the planning algorithm.

Regarding the works above, we see that EDA is only an alternative planning algorithm with a means to utilise prior knowledge when finding a solution plan and thus we do not aim to compare its performance with the algorithms in other approaches. Nevertheless, we share with them the common QoS attributes, but the reputation attribute is captured by a self-rating metric which is derived from a user's own experiences in the delivered QoS rather than from other users' subjective opinions. None of the related works address QoS-based service provision schemes and plan durability at planning time.

3 QoS-Based Service Provision Schemes and QoS Variability

In this section, we present our view on QoS-based service provision schemes and variability of the QoS towards plan durability. The following contribute to the extended QoS model and EDA-based planning in Sections 4 and 5.

3.1 QoS-Based Service Provision Schemes

Service providers can state the QoS in terms of complex service provision schemes to realise various classes of service provision. We give three examples here:

Multi-level QoS. A particular QoS attribute value may be published at different rates. Table 1 shows an example of multi-level availability of a service instance based on time of day. Multi-level QoS can be modelled in other ways, e.g. availability rates by classes of users, or by both time and classes of users. The scheme relating to cost, time, and reliability can be formulated in a similar manner.

Table 1. Example of multi-level availability

Service Instance A	
Time of Day	Availability
06:01 – 18:00	Base availability
18:01 – 00:00	+3 %
00:01 – 06:00	+5 %

Multi-level QoS affects service instance selection for abstract services. With the scheme in Table 1, a single service instance A effectively 'spawned' into three logical service instances; each of them is in service during particular time of day and with a particular availability rate. The planning algorithm considers them as three candidates for the abstract service.

QoS Interdependency. This service provision scheme forms a relation between different kinds of QoS attributes of a particular service instance. In Table 2, a service instance offers different classes of processing time based on service cost charged to the user. This QoS interdependency can be modelled in other ways, e.g. offering classes of discount for different levels of increased processing time. The scheme relating to availability and reliability can be formulated in a similar manner.

Table 2. Example of cost-processing time dependency

Service Instance A	
Cost	Processing Time
Base cost	Base processing time
+3%	-5%
+5%	-8%

Similarly to multi-level QoS, QoS interdependency affects the number of service instances that is associated with an abstract service. With the scheme in Table 2, a single service instance A is viewed as three logical instances; each of them offers service at the designated cost and processing time.

Partnership. Partnership refers to an agreement between service instances to offer a special class of service to attract users. The partnered instances may belong to the same service provider or different providers. A partnership scheme thus models dependencies between QoS attributes of the partnered instances. Table 3 shows a partnership scheme between service providers A, B, and C. The scheme offers a 10% discount in cost when the following instances of A, B, and C altogether participate in a particular plan: (1) any instance offered by A (2) any instance of abstract service X offered by B and (3) instance x1 of abstract service X or instance y1 of abstract service Y offered by C.

Partnership nevertheless can affect plan durability. When any two service instances are partners, it is likely that a QoS deviation in one instance could affect its partner. For example, when the instance y1 fails or has a QoS deviation and is replaced by another instance of a service provider D, the discount in Table 3 will no longer apply. The planning algorithm may choose to replace also the instances of A and B in order to benefit from the partnership scheme that D has with other service providers. In this view, partnership leads to coupling between service instances and the plan becomes less durable as a single deviation may incur change to a larger extent of the plan.

Table 3. Example of partnership

10% discount in cost when these instances collaborate	
Service Provider	Constraint on Instances
A	Any instance
B	Any instance of abstract service X
C	Instance x1 of abstract service X, or instance y1 of abstract service Y

3.2 QoS Variability

The QoS of a service instance may be affected not only by communication networks but also by the service instance itself. Since each service instance is built and tested independently in certain environment, the QoS behaviour may vary when it is used in different operational environment. Therefore different users may have different experiences in using the same service instance. We address QoS variability through user self-rating, which is given to service instances, and a supporting planning architecture.

Self-rating. Instead of using users' subjective opinions to determine the confidence in the overall quality of service provision, we aim for a self-rating approach which is more objective and respects users' personal experiences in service usage. Self-rating follows the idea of [14] such that service rating is based on deviation of the delivered QoS from the published QoS; the rating score is increased if the QoS fluctuates in a good way, and decreased otherwise. However, the score by [14] is computed at the service side and based on users' invocations from different network environments. The score is therefore biased from a particular user's viewpoint. We propose a self-rating metric (P) which reflects QoS fluctuations of a service instance experienced by a particular user:

$$Rating_s(P) = \frac{N}{E} \quad (1)$$

where N is the reward score given when the delivered QoS deviates in a good way, and E is the penalty score given otherwise. Rating runs between (0, 1], and when it reaches 1, it stops responding to any more rewards. For an initial rating given to any service instance that is first known to the user, we adopt the mid-value 0.5 rather than an external rating score (e.g. published rating or other users' rating). This is because we prefer the user to truly rate service behaviour from personal experiences and not to be biased by the score determined under different operational settings. This initial score, in other words, is a representation of an initial N (e.g. 10) divided by an initial E (e.g. 20). When a user invokes any service instance, delivered QoS will be measured in order to update rating according to the user's own rating rules. We allow for personal rating rules since different users may be sensitive to QoS deviation in different manners and may opt for different reward-penalty schemes. Table 4 shows rating rules for time and availability defined by a user. The time rating rules are based on the distance of the delivered QoS from the published QoS under an acceptable fluctuation range (± f). The availability rating rules penalise the service instance if it is not accessible at the time the invocation is made and retried. Given a scenario that a service instance is known to a user for the first time, the user sees the rating score 0.5 (i.e. 10/20). Suppose when the user invokes the service instance, it does not respond at first but a retry succeeds, and the delivered time of this invocation falls under the third rule of time rating rules. Hence the score of the service instance in this scenario will be (10+1)/(20+1) = 11/21 = 0.524. The values N = 11 and E = 21 become the new base values for this service instance for the next rating computation. While the user experiences the quality of the service instance through repeated invocations, the rating score is refined and becomes more accurate.

Table 4. Example of personal rating rules for time and availability

Time		Availability	
Event	Action	Event	Action
• $T_s$delivered > $T_s$published + f	+1 to *E*	• Not available first time	+1 to *E*
• $T_s$published + f ≤ $T_s$delivered ≤ $T_s$published – f	+0.25 to *N*	• Not available next time	+2 to *E*
• $T_s$delivered < $T_s$published – f	+1 to *N*		

This QoS-based self-rating contributes to plan durability. If all service instances in a plan are quite stable or do not fluctuate much in a bad way (i.e. good rating), the plan becomes durable and can be reused. On the contrary, if any service instance behaves much badly (i.e. low rating), the QoS of the whole plan may be affected and replanning becomes necessary.

Planning Architecture. To support QoS-based service provision schemes and QoS variability, we assume each user has a planning architecture as in Fig. 1.

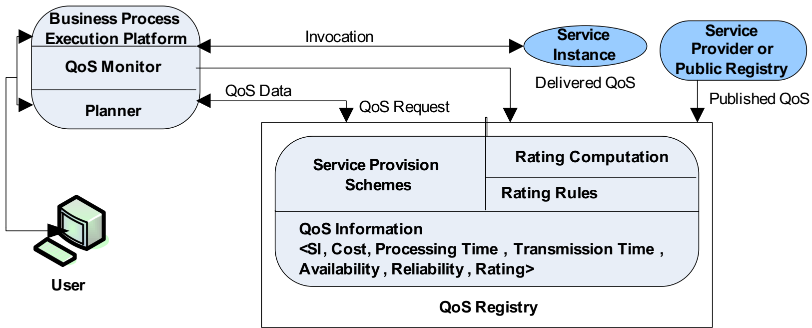

Fig. 1. Client-side planning architecture

The QoS registry stores QoS information of all service instances which are relevant to his/her business domain; discovery of these instances can be performed manually or automatically prior to composition. Cost, time, availability, and reliability information is retrieved from a public service registry or directly from service providers, and can be refreshed periodically or before planning. In contrast, rating information is initialised to 0.5 and gets updated only by rating rules. Note that each of the logical instances (e.g. each of the three logical instances of A according to Table 1) has its QoS information stored separately but they all share the same rating score. With the QoS information, plans are created by the planner and executed on the business process execution platform. During execution, delivered QoS is monitored by the QoS monitor and fed to the QoS registry where rating is then computed for service instances. Such knowledge of the QoS and personal experiences regarding particular service instances can help the planner to replan existing abstract flows for the user when necessary and to compose plans for new abstract flows that involve those service instances.

4 Extended QoS Model

Our QoS model comprises five quality attributes: time (i.e. processing time + transmission time), cost, availability, reliability, and rating. The definitions of the first four can be found in [2] while rating refers to the proposed self-rating in Section 3.2. Since a composition can be viewed as an aggregation of control flow constructs, the overall QoS of the flow is based on the QoS concerning each construct. We adopt a set of QoS metrics for common control flow constructs (i.e. sequence, switch, fork, and loop) of [8] and extend it with self-rating metrics (shown in boldface type) as in Table 5.

Table 5. Metrics for control flow construct-QoS pairs

QoS	Sequence	Switch	Fork	Loop
Time (T)	$\sum_{i=1}^{m} T(t_i)$	$\sum_{i=1}^{n} p_{ai} * T(t_i)$	$Max\{T(t_i)_{i\in\{1...p\}}\}$	$k * T(t)$
Cost (C)	$\sum_{i=1}^{m} C(t_i)$	$\sum_{i=1}^{n} p_{ai} * C(t_i)$	$\sum_{i=1}^{p} C(t_i)$	$k * C(t)$
Availability (A)	$\prod_{i=1}^{m} A(t_i)$	$\sum_{i=1}^{n} p_{ai} * A(t_i)$	$\prod_{i=1}^{p} A(t_i)$	$A(t)^k$
Reliability (R)	$\prod_{i=1}^{m} R(t_i)$	$\sum_{i=1}^{n} p_{ai} * R(t_i)$	$\prod_{i=1}^{p} R(t_i)$	$R(t)^k$
Rating (P)	$\prod_{i=1}^{m} \boldsymbol{P}(t_i)$	$\sum_{i=1}^{n} p_{ai} * \boldsymbol{P}(t_i)$	$\prod_{i=1}^{p} \boldsymbol{P}(t_i)$	$\boldsymbol{P}(t)^k$

The metrics are recursively defined on compound nodes of the flow. For a Sequence construct of tasks $\{t_1,..., t_m\}$, the time and cost metrics are additive, while availability, reliability, and rating are multiplicative. Each of the Cases 1,…, n of the Switch construct is annotated with the probability to be chosen (p_{ai}); probabilities are initialised by the user and can be updated later considering the information obtained by monitoring flow execution. The functions for the Fork construct are essentially the same as those for the Sequence construct, except for the time attribute where this is the maximum time of the parallel tasks $\{t_1,..., t_p\}$. Finally, the Loop construct with k iterations of task t is equivalent to the Sequence construct of k copies of t.

5 Planning Algorithm

Planning a composite service is a constraint optimisation problem that needs to

1. Meet user QoS constraints. For example, an abstract service must not have service cost above a given limit, or the overall cost of the plan is constrained. The former is called a local constraint and the latter a global constraint.
2. Optimise a function of some QoS attributes. For example, the user may want to minimise service time while keeping cost below the limit.

This section describes how EDA is applied to find QoS-optimised solution plans.

5.1 Planning with EDA

Like other evolutionary computation techniques, EDA follows the process in Fig. 2(a) to find a solution to an optimisation problem.

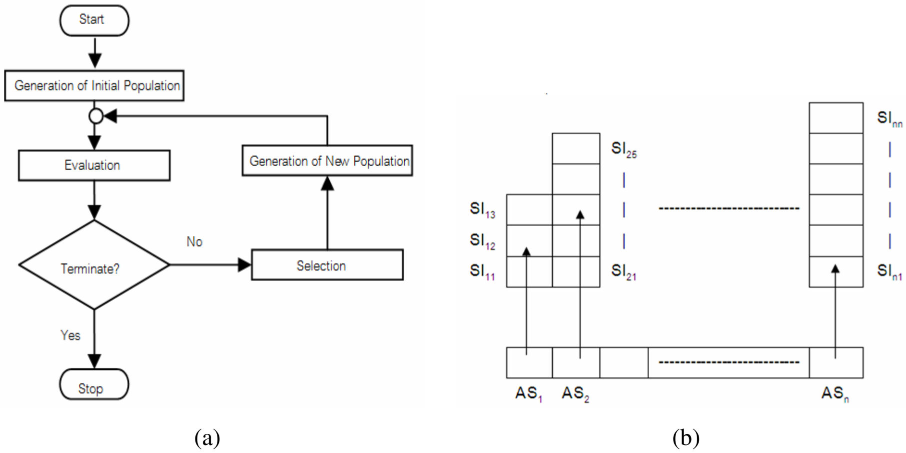

Fig. 2. EDA (a) Evolutionary computation process (b) Chromosome encoding (similar to encoding in GA [8])

The algorithm starts with a generation of a fixed-size initial population, which consists of a number of randomised encoded solutions called chromosomes. The initial population is allowed to evolve under specified selection rules to a state that satisfies user constraints and optimises a particular fitness function. Each chromosome will be evaluated against the constraints and its fitness is computed. If any chromosome satisfies the constraints, the algorithm may stop and the chromosome becomes the solution. Otherwise the algorithm continues to find a more optimised solution until a certain number of generations have been processed. To generate a new generation of population, some best-fitted chromosomes from the previous generation are selected for the new generation, and additional chromosomes are generated until the population size is reached. Then the algorithm repeats.

An EDA chromosome is encoded in a bit string of a fixed length. Sub-bit strings represent service instances (SI) that are mapped to abstract services (AS). In Fig. 2(b), AS_1 has three *SI*s and therefore is encoded with two bits, whereas AS_2 has five *SI*s and is encoded with three bits. Suppose SI_{12} is selected for AS_1, its sub-bit string is *10*, and if SI_{25} is selected for AS_2, its sub-bit string is *101*. A chromosome is then a sequence of sub-bit strings representing all selected service instances for the abstract flow.

Each chromosome g in a generation has its fitness computed by using the following fitness function (to be minimised); the function is similar to the one proposed in [8] except for $Rating(g)$ and $D_p(g)$ components (shown in boldface type) that we augment to represent self-rating and partnership coupling respectively:

$$F(g) = \frac{w_1 Cost(g) + w_2 Time(g)}{w_3 Availability(g) + w_4 Reliability(g) + \boldsymbol{w_5 Rating(g)}} + w_6 D(g) + \boldsymbol{w_7 D_p(g)} \quad (2)$$

- *Cost*(g), *Time*(g), *Availability*(g), *Reliability*(g), and *Rating*(g) are the QoS values computed for the chromosome g using the metrics in Table 5.
- $D(g)$ is the distance of the chromosome g from constraints satisfaction, i.e. $F(g)$ penalises the chromosome that does not meet the user's constraints and drives the evolution towards constraints satisfaction. It is defined by

$$D(g) = \sum_{i=1}^{n} cl_i(g) * y_i$$

  where $cl_i(g)$, $i = 1,\ldots, n$, is g's distance from i^{th} constraint, and

$$\begin{cases} y_i = 0 & \text{when } cl_i(g) \le 0 \text{ (positive or no distance)} \\ y_i = 1 & \text{when } cl_i(g) > 0 \text{ (negative distance)} \end{cases}$$

- $D_p(g)$ is the degree of partnership coupling, i.e. $F(g)$ penalises the chromosome in which associated service instances are part of partnership schemes. It is defined by

$$D_p(g) = \frac{S_p}{n}$$

  where S_p is the number of service instances involved in partnership schemes, and n is the total number of service instances in g.
- w_i indicates the weight (i.e. importance) the user gives to each component of $F(g)$.

When a satisfactory solution is not yet found, there are several strategies for EDA to generate a new generation of population. We use the one called Probabilistic Building Increasing Learning (PBIL) to generate chromosomes for the new generation by using Generator Function (GF). A GF contains probabilities p_i, i.e. $\{p_1, \ldots, p_n\}$, where p_i is the probability that the ith bit of an n-bit chromosome is 0. For a given population with m chromosomes, p_i is the proportion of the number of 0 bits found in the ith bit position to the total number of bits in the ith bit position (i.e. m). For example, given a population with four chromosomes {*010, 100, 111, 101*}, the GF contains $p_1 = 0.25$, $p_2 = 0.5$, and $p_3 = 0.5$. To generate a new generation of population, each new chromosome in the new population would have 0 (zero) assigned to its 1st, 2nd, and 3rd bit with the probabilities 0.25, 0.5, and 0.5 respectively. In this manner, GF reflects knowledge from the past which guides how to generate good chromosomes. This knowledge would be refined as the population evolves from one generation to the next.

5.2 Durable Planning

In the fitness function above, rating and partnership coupling components contribute to durability of the generated plan. Since rating concerns QoS fluctuation while partnership coupling signifies a potential that a single service change may affect the plan to a larger extent, putting weights on them will indicate to EDA to find an optimised

plan with good rating and low partnership coupling. That is, when it is less likely for the plan to require change, the plan is durable and can be reused.

At execution time, service instances, and hence the business process flow, may suffer from performance degradation and cannot deliver service quality as planned. The flow should be prepared to survive in unstable operational environment by considering performance deviation at planning time. We can simulate the situation by injecting QoS deviation to service instances and letting the EDA process makes a plan out of those instances (see Section 6.2).

6 Experimental Studies

We conducted a couple of simulations to study the behaviour of EDA-based planning with respect to QoS-based service provision schemes and plan durability. Note that service provision schemes took part in the experiments by constraining the QoS of candidate service instances. The first study focused on the use of GF from previous planning in building a new composition plan when published QoS of service instances was updated. The second study focused on durable planning. In each study, the population size was 200, the maximum number of generations to run EDA is 50, and the experiment was repeated for 50 times to obtain average results. The simulation program was written in Java with J2SDK 1.6. Experiments were run on a 1.8 GHz Intel Pentium^{TM}, 1 GB of RAM, and Ubuntu Linux version 7.04.

6.1 Use of GF

This study focused on the use of GF from previous planning in building a new composition plan when the published QoS of service instances was updated by service providers. This will demonstrate how GF benefits a search for a new solution plan. Suppose a user constraint was that the fitness value of the plan had to be below 9,600. The QoS of service instances was updated 4 times after the instances were first published. To simulate each update, we degraded all QoS values of the instances 0-5% at random. For example, if a service instance, with 1,500-millisecond service time, was randomised to degrade 1%, its service time would be updated to 1,515 milliseconds. At each update, EDA generated a new plan. There were 10 abstract services and each of them had 111 candidate service instances.

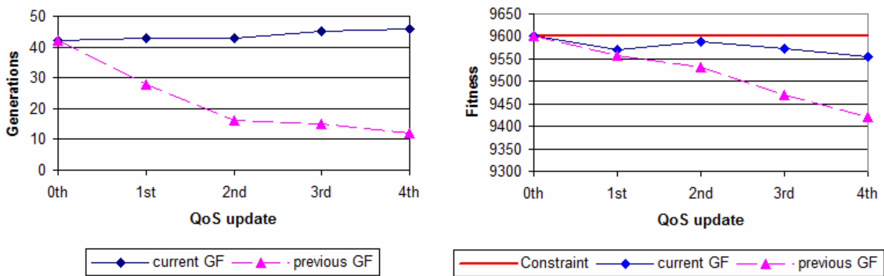

Fig. 3. Composition with GF

We compare between a composition that does not use GF from previous planning (i.e. it uses GF in current planning only) and the one that does. In Fig. 3, the left graph shows that, when GF from previous planning involves in the creation of chromosomes in the new planning after a QoS update, convergence time decreases. This means it takes less time (i.e. less number of generations) to find an optimised solution plan. This is because GF from previous planning is the knowledge that guides the characteristics of good solutions. It can be seen that, for example, the 1st – 50th generations of the planning on the 1st QoS update are effectively the 51st – 100th generations of the initial 0th planning. The right graph shows that, under a user constraint on the fitness value of the plan (i.e. below 9,600), composition that uses GF from previous planning gives better solutions. As GF is passed along, each composition gives a more optimised solution plan.

6.2 Plan Durability

Usually service QoS that is published by service providers is considered during planning. We expect that early (i.e. planning-time) consideration about the possibility of QoS deviation from what was published should result in solution plans that are more durable at execution time. This study focused on composition of plans that can survive unstable execution environment. Given a flow of 20 abstract services and a user constraint such that the fitness value of the plan had to be under 9,600, two groups of 100 plans were generated. For the first group, published QoS was considered during planning; this represented composition with ideal service instances with no QoS deviation. The second group comprised the concrete plans from the first group but with degraded QoS; this represented composition with an expectation of service QoS deviation. The service instances of each plan within the second group had all their QoS values degraded by 1-5% randomly.

After two groups of 100 concrete plans were obtained, we simulated their execution under unstable environment. Each service instance in any of these plans was randomised with a 40% chance to have its QoS degraded at run time. For the service instance that was to degrade, its QoS was degraded by 1-10% randomly. Then the fitness values of the plans in these two groups were computed to determine a percentage of survival, i.e. how many of the plans in each group still met the user constraint (with an acceptable 5% deviation) in unstable runtime environment.

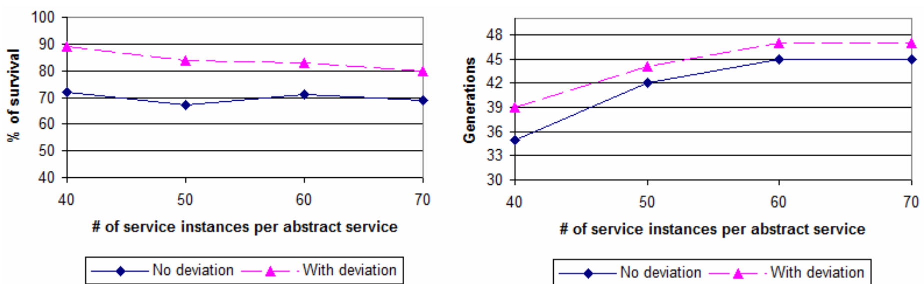

Fig. 4. Effects of plan durability

We experimented with various number of service instances per abstract service; this reflected the variation in size of solution search space. In Fig. 4, the left graph shows that the plans in the second group (i.e. those created with consideration about QoS deviation) can survive runtime degradation better than those in the first group (i.e. those created with no QoS deviation). This observation is true regardless of the size of solution search space. Nevertheless, the right graph indicates that it takes longer time (i.e. more number of generations) for EDA to find an optimised plan when the possibility of QoS deviation is considered during planning.

7 Discussion and Conclusion

In this paper, we discuss various schemes of QoS-based service provision and address a plan durability issue concerning QoS deviation and service relations. Self-rating and partnership coupling are introduced as part of the extended QoS model for service instances and workflows. A client-side planning architecture is also proposed. Using EDA as the planning algorithm, our experiments show that GF can benefit planning since knowledge of good solutions is utilised in finding a QoS-optimised plan. Considering the possibility of QoS deviation early at planning time will also result in more durable plans which can survive performance degradation at execution time.

On plan durability, our approach does not aim to make very durable plans so as to replace runtime replanning. Replanning capability is necessary when new services are offered or there are service outages or severe QoS degradation during flow execution. In commercial scenarios today, it is common practice that service providers publish their service instances QoS with a possible reduced QoS rate as a safety buffer. This safety buffer is taken into consideration at run time to determine QoS violation. Our work is aligned with this compromised QoS approach but takes the QoS safety buffer into consideration at planning time. By planning with degraded service instances in mind, we obtain the solutions that are more durable at run time. Our approach thus reduces the chance that a solution will need runtime replanning. It is also worth noting that our approach assumes the published QoS information is accurate. If service providers understate their service QoS only to boost their ratings, they put their service instances at the risk of not being selected to the plans from the beginning.

On performance of EDA, we rely on the performance result of the GA-based algorithm compared to that of the integer programming approach as reported in [8]. GA takes less time to find a solution and its timing performance is almost constant when the solution search space grows (i.e. when the number of service instances per abstract service increases). Thus it is preferred for the case of widely used abstract services, such as hotel booking and ecommerce services, which have a large number of candidate service instances. By using EDA, we also observe that the solutions generated in each generation can be very much similar to those in the previous generation because of knowledge in GF. That is, GF can lead EDA to fall easily into local optima. We will find a way to detect the situation and adjust GF. Nevertheless, we expect that knowledge in GF would be useful for runtime replanning, either in making a whole new plan or replacing specific part of the plan. We will explore more about the influence of GF over replanning.

References

1. Barry, D.K.: Web Services and Service-Oriented Architecture. Morgan Kaufmann, San Francisco (2003)
2. Zeng, L., Benatallah, B., Ngu, A.H.H., Dumas, M., Kalagnanam, J., Chang, H.: QoS-Aware Middleware for Web Services Composition. IEEE Transactions on Software Engineering 30(5), 311–327 (2004)
3. Cardoso, J., Sheth, A., Miller, J., Arnold, J., Kochut, K.: Quality of Service for Workflows and Web Service Processes. Journal of Web Semantics 1(3), 281–308 (2004)
4. Menasce, D.A.: QoS Issues in Web Services. IEEE Internet Computing 6(6), 72–75 (2002)
5. Mani, M., Nagarajan, A.: Understanding Quality of Service for Web Services, `http://www-128.ibm.com/developerworks/library/ws-quality.html`
6. Goldberg, D.E.: The Design of Innovation Lessons from and for Competent Genetic Algorithms. Kluwer Academic Publishers, Dordrecht (2002)
7. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
8. Canfora, G., Di Penta, M., Esposito, R., Villani, M.L.: An Approach for QoS-Aware Service Composition Based on Genetic Algorithms. In: Proceedings of Genetic and Evolutionary Computation Conference (GECCO 2005), Washington DC, USA, June 2005, pp. 1069–1075 (2005)
9. Chang, W.C., Wu, C.H., Chang, C.: Optimizing Dynamic Web Service Component Composition by Using Evolutionary Algorithms. In: Proceedings of 2005 IEEE/WIC/ACM International Conference on Web Intelligence September 2005, pp. 708–711 (2005)
10. Wu, B.Y., Chi, C.H., Xu, S.: Service Selection Model Based on QoS Reference Vector. In: Proceedings of 2007 IEEE Congress on Services (SERVICES 2007), Salt Lake City, Utah, July 2007, pp. 270–277 (2007)
11. Grønmo, R., Jaeger, M.C.: Model-Driven Methodology for Building QoS-Optimised Web Service Compositions. In: Kutvonen, L., Alonistioti, N. (eds.) DAIS 2005. LNCS, vol. 3543, Springer, Heidelberg (2005)
12. Cardellini, V., Casalicchio, E., Grassi, V., Lo Presti, F.: Flow-Based Service Selection for Web Service Composition Supporting Multiple QoS Classes. In: Proceedings of 2007 IEEE International Conference on Web Services (ICWS 2007), Salt Lake City, Utah, July 2007, pp. 743–750 (2007)
13. Aggarwal, R., Verma, K., Miller, J., Milner, W.: Constraint-Driven Web Service Composition in METEOR-S. In: Proceedings of IEEE International Conference on Services Computing (SCC 2004), Shanghai, China, September 2004, pp. 23–30 (2004)
14. Mourad, O., Athman, B.: Efficient Access to Web Services. IEEE Internet Computing, 34–44, March-April (2004)