JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 30, 957-972 (2014)

Dynamic Service Composition Using Core Service Identification*

SHANG-PIN MA[1], YONG-YI FANJIANG[2] AND JONG-YIH KUO[3]
[1]*Department of Computer Science and Engineering*
National Taiwan Ocean University
Keelung, 202 Taiwan
[2]*Department of Computer Science and Information Engineering*
Fu Jen Catholic University
New Taipei City, 242 Taiwan
[3]*Department of Computer Science and Information Engineering*
National Taipei University of Technology
Taipei, 106 Taiwan

Service selection and binding for building composite services is a critical but difficult to resolve issue in the domain of service-oriented computing. This paper proposes an innovative approach to address this issue, in which every abstract component service in the composition is assigned a weight value to represent its importance according to designer scoring and basis path analysis, thereby generating a weighted service flow. Composite services, including the selected concrete component services, are produced according to the weighted service flow and genetic algorithms. The generated composite service ensures acceptable quality level and guarantees that the core component services, which are most likely to influence the overall service flow, are more robust than the others. Besides, a novel fault handling method based on the identified core services is proposed to enhance the availability of composite services. Experimental results demonstrate that the proposed approach can effectively maintain the quality of core component services.

Keywords: service-oriented architecture, service composition, quality of service, service optimization, genetic algorithm

1. INTRODUCTION

Service-Oriented Computing (SOC) has become an important trend in software engineering, exploiting both web services and Service-Oriented Architecture (SOA) as fundamental elements in the development of on-demand applications. Web services are self-described, self-contained, and platform-independent computational elements that can be published, discovered, and composed using standard protocols to build applications across various platforms and organizations. Recently, considerable researches have been conducted in both industry and academia to overcome various obstacles related to service-oriented technologies, such as semantic service discovery, automatic service composition, and QoS-aware service management.

It is widely accepted [1-3] that combining multiple web services into a composite service is more beneficial to users than finding a complex and preparatory atomic service that satisfy a special request. The resulting composite services can be used as atomic

Received March 6, 2013; revised August 6, 2013; accepted September 30, 2013.
Communicated by Jonathan Lee.
* This research was sponsored by National Science Council, Taiwan under the Grant NSC 102-2221-E-019-024.

services themselves in other service compositions to satisfy clients' requests. Generally, composite service refers to a service flow comprising abstract component services (often called service nodes or service tasks). Concrete component services corresponding to an abstract service are functionally equivalent, and may be substituted for one another. Most importantly, concrete services can be dynamically bound within a composite service to construct an executable service. Determining the means to select appropriate concrete component services for the generation of an executable composite service is of critical importance; however, it poses a number of challenges.

Quality of Service (QoS) is always an important concern in the selection of services. QoS can be represented by the attributes of a service, such as response time, price, availability, and reliability, and users can specify QoS constraints or preferences to influence the results of service selection. A number of approaches have been proposed to realize QoS-aware service selection and composition. These works utilized techniques of integer programming [4-7], AI planning [8], graph search [9, 10], and evolutionary computing [11-14] to find optimal service compositions from among a lot of candidates. In this paper, we argue that not all component services in a composition are equally important, while current efforts do not take this issue into consideration. To make an analogy, in real world, the engine of a car is regarded as a core component, because a failure of the engine results in a total failure of all operations within the car. The engine must be carefully maintained if the car is to remain operable. Similarly, taking another example in the field of software, the functions of flight reservation and hotel booking are treated as core components in a travel planning application, because failures of these functions may cause that other auxiliary functions, such as car rental or searching for nearby restaurants/scenic spots, are unable to operate or unable to generate required information. The idea of core component can also be applied to service composition technology. It is important to identify core component services (abbreviated as "core services") within a composite service, maintain the quality of these services, and achieve fault-tolerance or error recovery when unexpected situations arise. Although the notion of core service is not brand new [15], solving QoS-driven service optimization issue by applying core service concept is a novel but reasonable attempt. Besides, the service selection problem for service composition has been proven as NP-Complete since there are n^m combinations for a given composite service involving m tasks with n candidate composite service for each task [16].

To address the service selection issue, evolutionary algorithms are required to find solutions of service combinations from numerous services. Accordingly, this study proposes a methodology based on genetic algorithms for the selection of concrete component services. The overview of this work is shown in Fig. 1. There are three main phases involved: design phase, composition phase and execution phase. In the design phase, the process designer (*i.e.* the composite service provider) prepares the abstract composite service flow firstly, and then the proposed approach assigns every abstract component service in the composition a weight value to represent its importance according to designer scoring and basis path analysis, and thereby generate a weighted service flow. In the composition phase, the parameters of the proposed fitness function are set according to user preferences and the generated weighted service flow. The concrete composite service, including the selected concrete component services, is produced by performing genetic algorithms. In the execution phase, the concrete composite service is executed by

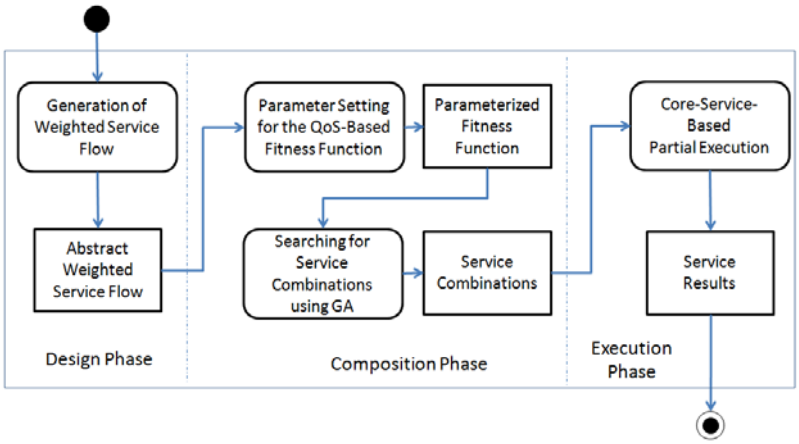

Fig. 1. Overview of the proposed approach.

applying the proposed "partial execution" mechanism to ensure the successful execution of core services and to prepare available service results when any service fault occurs. In summary, the proposed mechanism enables the generation of executable composite services with robust core component services to ensure an acceptable level of quality.

The contributions of this work can be stated as follows.

– Integrating the notion of core components with the web service composition and optimization techniques.
– Providing a systematic approach to find near-to-optimal service compositions with robust core component services.
– Furnishing a novel fault handling method based on the identified core services to enhance the availability of composite services.

This paper is organized as follows. Section 2 outlines previous research related to this study. Section 3 describes the proposed approach to the selection of services for a composition. Section 4 presents experimental data supporting the proposed approach and discusses possible threats. Conclusions and future work are presented in Section 5.

2. RELATED WORK

In this section, we introduce a number of studies related to the dynamic composition for web services. There are three main research tracks for addressing this issue: optimization through integer programming, AI planning or graph search, and evolutionary-computing-based approaches.

Applying integer programming to solve the optimization issue of dynamic service composition is an important research trend. Zeng *et al.* [4, 17] presented a QoS model to define the measurement of non-functional quality associated with composite services and proposed an approach for finding the optimal composition by mixed integer programming (MIP). Aggarwal *et al.* [5] proposed a framework for constraint-driven service composition, providing a tool that composes web services by considering QoS attributes

and applying an Integer Programming Solver call LINDO. This work is a part of METEOR-S (Managing End-To-End OpeRations) project, which focuses on use semantics for the lifecycle of semantic web processes. Qi *et al.* [6] proposed a heuristic service composition method named LOEM (Local Optimization and Enumeration Method) to find near-to-optimal compositions by filter out non-promising candidate services for each task in the composition. This work also transforms the service composition problem into a MIP one to find solutions. Alrifai *et al.* [7, 18] presented a hybrid solution that combines global optimization with local selection techniques to address the performance issue of normal global optimization for service composition. They used MIP to find the optimal decomposition of global constraints into local constraints and to find the best services that satisfy the local constraints. Because this work does not require to import all QoS data of available services while performing MIP, it can reduce the overhead of finding a combination of services and achieve close-to-optimal results.

The problem of dynamic service composition can be transformed as a planning problem or a graph search problem. Wu *et al.* [8] presented an AI planning approach for the automatic composition of web services. In their approach, planning is generated by extracting web service descriptions in SAWSDL (Semantic Annotations for WSDL), extending pre-conditions and post-conditions, extending states by adding sets of data types, and modeling the requirement of the service requester using Semantic Template. Finally, a context-based ranking algorithm is employed to select the best element from among the available candidates. Jiang *et al.* [9] developed a tool, named QSynth, for the automatic composition of QoS-Aware services, viewing it as a graph search problem. After building a plan to address a request, the final solution is determined by identifying sub-graphs that provide the global optimal QoS. They defined four types of QoS measures, including sum type (such as response time), min type (such as throughput), multiplication type (such as reputation), and max type, with three composition patterns: sequence, joint and split. Different patterns require different computing rules, and the QoS computing algorithm focuses on single QoS measurements. Zheng *et al.* [10] designed a general algorithm for QoS calculation of composite services with complex structures. They modeled the composite service as a directed graph with arbitrary composition patterns. de Oliveira Jr. *et al.* [19] presented a heuristic-based algorithm for dynamic service composition by extending the method proposed by Weise *et al.* [20]. This work devised a set of heuristic rules in the proposed comparator function to consider five composition properties: known concepts, unknown concepts, eliminated concepts, the number of services, and the overall QoS score, while choosing solutions from among candidate compositions.

Numerous efforts utilize techniques of evolutionary computation to solve the issue of finding service combinations. Canfora *et al.* [21] proposed an approach to the problem of service composition optimization using GA. By comparing performance and scalability with increasing numbers of candidate web services and tasks, they demonstrated that GAs outperform linear integer programming. Canfora *et al.* also proposed a re-planning approach based on GA [11]. When the actual QoS deviates from the estimation during the execution of workflow, the re-planning approach determines the re-planning slice and re-binds the concrete services. Ye & Mounla [12] proposed a hybrid approach combining Integer Programming (IP), a genetic algorithm, and case-based reasoning to address the problem of QoS-aware service composition. Fanjiang *et al.* [13] also provided

an approach to support constructing dynamic service workflow according to the user's requirements based on genetic algorithms and case-based reasoning. Rosenberg *et al.* [14] presented a metaheuristic framework to address QoS-aware web service compositions in large-scale service-oriented systems. Their approach provided an extensible QoS model focusing on operational QoS attributes (*e.g.*, response time, and business-related attributes (*e.g.*, price) and implementing three algorithms of metaheuristics, namely GA (genetic algorithms), SA (simulated annealing), and TS (tabu search), to find optimal solutions for the composition of web services. Both of the studies conducted by Aversano *et al.* [22] and Gao *et al.* [23] presented genetic algorithms with tree-coded structures to model workflow patterns in composite services. Our approach also applies genetic algorithms; however, we integrated the concept of core services into the service selection strategy to produce a more robust service combination.

3. APPROACH DESCRIPTION: CORE-SERVICE-BASED SERVICE COMPOSITION

This section presents the main methods in the proposed approach. To illustrate these mechanisms, we have provided a motivating example of an application designed to assist travelers (see Fig. 2). This example presents a composite service comprising 10 component services (more precisely, 10 service operations). We have also provided a sequence number for each component service to facilitate the explanation.

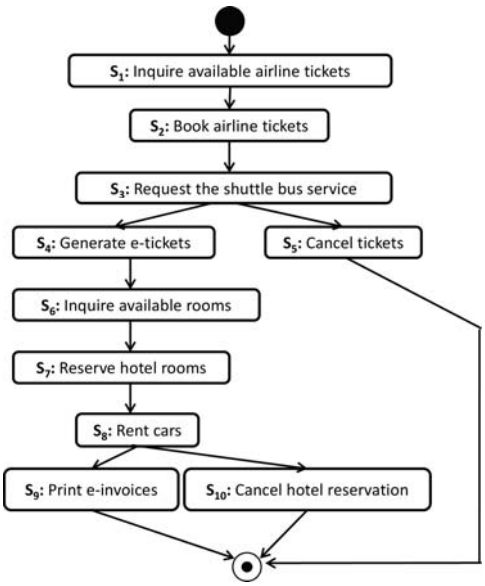

Fig. 2. Example services associated with travel (travel service).

The proposed process to design and operate composite services can be divided into four parts:

1. Generation of weighted service flow

2. Parameter setting for the proposed fitness function
3. Application of genetic algorithms for finding optimal service compositions
4. Execution of composite services with the proposed fault handling mechanism called partial execution

Significant parts are detailed in the following subsections.

3.1 Generation of Weighted Service Flow

We propose an analytical approach to the generation of a weighted service flow, based on values assigned by the process designer and the results of execution path analysis.

3.1.1 Assigning importance scores by process designers

We assumed that the process designer would be able to determine, in advance, the initial importance of each service node in the workflow. The process designer assigned an Importance Point (IP) to each node, ranging from 1 to 10 with a higher score indicating that the corresponding component service is more important than the others. We then converted all scores to reduce their sum to 1. The weight W_u ("u" indicates "user") of each service node from the viewpoint of the process designer can be calculated:

$$W_u(S_n) = \frac{IP(S_n)}{\sum_k IP(S_k)}. \quad (1)$$

In the motivating example, booking airline tickets and reserving hotel rooms is the ultimate goal. Accordingly, the process designer assigned an importance of 10 points (Importance Point, IP) to the core services of "book airline tickets" and "reserve hotel rooms", while the others services were assigned scores 1 point each. The weight of each service was then computed using Eq. (1). The W_u scores of "booking airline tickets" and "reserve hotel rooms" were both 0.357, and the remaining W_u scores were 0.036.

3.1.2 Execution path analysis

We leveraged the basis path testing method [24], to retrieve all possible representative execution paths for the composite service. In the example in Fig. 2, three basis paths were identified:

1. S_1–S_2–S_3–S_5
2. S_1–S_2–S_3–S_4–S_6–S_7–S_8–S_9
3. S_1–S_2–S_3–S_4–S_6–S_7–S_8–S_{10}

According to these basis paths, some of the service nodes played more important roles than others, because they emerged in more basis paths. Thus, we assigned an Execution Point (EP) score to each service node based on their probability of emergence:

$$EP(S_n) = \frac{BP(S_n)}{BP} \quad (2)$$

where $BP(S_n)$ is the number of basis paths that include service node n, and BP is the number of all basis paths.

In the example in Fig. 2, we obtained the following results via execution path analysis:

- Score of S_1, S_2, S_3 is 1
- Score of S_4, S_6, S_7, S_8 is 2/3
- Score of S_5, S_9, S_{10} is 1/3

We converted all of the scores to reduce the sum to 1. The weight W_e ("e" indicates "execution path") of each service node from the viewpoint of path analysis can be computed as:

$$W_e(S_n) = \frac{EP(S_n)}{\sum_k EP(S_k)}. \tag{3}$$

3.1.3 Calculation of final weight values

We provided two kinds of weights for the component services to represent the viewpoints of the process designer and path analysis. We combined these two values into the following formula:

$$W(S_n) = SN * (I_u * W_u(S_n)) + (1 - I_u) * W_e(S_n)) \tag{4}$$

where $0 \le I_u \le 1$, and SN is the number of all service nodes.

On the basis of the final weight value, a weighted service flow can be produced. The weighted service flow of the example is shown in Fig. 3. (Here assign I_u as 0.7 since we lay more stress on the designer's viewpoint)

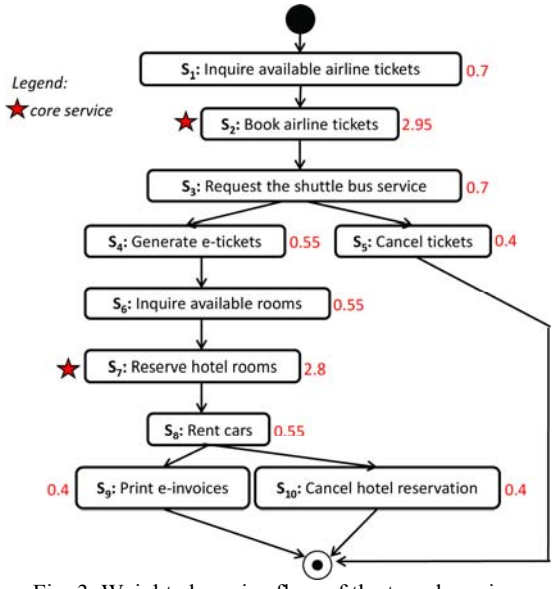

Fig. 3. Weighted service flow of the travel service.

3.2 Parameter Setting for the Proposed Fitness Function

Because QoS is a significant factor in the selection of concrete services, we sought to enhance the aggregation functions proposed by [11, 21]. The extended QoS aggregation functions are shown in Table 1. The weight value of each component service in the weighted service flow was integrated into the functions. When calculating the QoS value of time, weight values are a factor in the formula; however, for availability and reliability, weight values are an exponent. Notably, the aggregation function used to calculate total price keeps the original definition because the price of core services does not influence the execution of composite services.

Table 1. QoS aggregation functions.

QoS Attr.	Sequence	Switch	Flow	Loop
Time	$\sum_{i=1}^{n} T(Si)\cdot W(Si)$	$\sum_{i=1}^{n} Pa_i\cdot T(Si)\cdot W(Si)$	$Max\{T(Sj)_{j\in\{1\ldots n\}}\}\cdot W(Si)$	$k*T(Si)\cdot W(Si)$
Reliability	$\prod_{i=1}^{n} R(S_i)^{W(Si)}$	$\sum_{i=1}^{n} Pa_i\cdot R(S_i)\cdot W(S_i)$	$\prod_{i=1}^{n} R(S_i)^{W(Si)}$	$R(S_i)^{W(Si)\cdot k}$
Availability	$\prod_{i=1}^{n} A(S_i)^{W(Si)}$	$\sum_{i=1}^{n} Pa_i\cdot A(S_i)\cdot W(S_i)$	$\prod_{i=1}^{n} A(S_i)^{W(Si)}$	$A(S_i)^{W(Si)\cdot k}$
Price	$\sum_{i=1}^{n} P(Si)$	$\sum_{i=1}^{n} Pa_i\cdot P(Si)$	$\sum_{i=1}^{n} P(Si)$	$k\cdot P(Si)$

Pa: probability
K: the number of iterations

Service requesters may select the degree of importance for each QoS factor by assigning four weights, w_1, w_2, w_3 and w_4, representing the importance of availability, reliability, response time, and price, respectively. For example, if the user attaches great importance to the price of the composite service, w_4 can be set as a large value and other weights can be set as relatively small values.

Because we adopted genetic algorithms (GA) to search for a solution for the composite service (*i.e.* combination of concrete services), it was necessary to encode the chromosome. In our definition, a chromosome is an array with several items; each item representing an abstract component service in the service flow referring to a concrete component services that matches the interface of the abstract service. A combination of concrete component services belonging to different items is a solution of the service composition. The fitness function $F(c)$ for chromosome c is defined below:

$$F(c)=\frac{w_1 * \text{Aggregated Availibilty}(c)+w_2 * \text{Aggregated Availibilty}(c)}{w_3 * \text{Aggregated Time}(c)+w_4 * \text{Aggregated Time}(c)}. \quad (5)$$

In the fitness function, all QoS values are calculated from the function definitions in Table 1. This maximizes availability and reliability, and minimizes time and price within

the constraints specified by the process designer.

We have provided another simple example to illustrate the effect of the weighted QoS formula (see Fig. 4). The structure of the illustrative composite service comprises five component services. The weight value of core service S_3 is 0.5.

$\sum_{i=1}^{n} T(S_i) \cdot W(S_i)$

CASE-1 : core service s_3 has better QoS of response time

service	time1	weight	weighted time
s_1	0.8	0.1	0.08
s_2	0.5	0.1	0.05
s_3	0.3	0.5	0.15
s_4	0.9	0.2	0.18
s_5	0.6	0.1	0.06
total	3.1	1	0.52

CASE-2 : core service s_3 has worse QoS of response time

service	time2	weight	weighted time
s_1	0.6	0.1	0.06
s_2	0.5	0.1	0.05
s_3	0.9	0.5	0.45
s_4	0.8	0.2	0.16
s_5	0.3	0.1	0.03
total	3.1	1	0.75

Fig. 4. Verifying the influence of the weighted formula.

We discuss two cases of execution time. In Case 1, the core services require less execution time (0.3), and the overall time of the composite service is 3.1; in Case 2, the time required for the execution of core services (0.9) is worse than that of Case 1; however, the composite services require the same overall time. After the weighted calculations, the overall time score of Case 1 (0.52), is better than that of Case 2 (0.75). Accordingly, the composite service with better quality core services will be chosen.

3.3 Application of Genetic Algorithms for Finding Optimal Service Compositions

In our GA-based search approach, the chromosome is an array representing the service composition and comprising multiple selected component services. The fitness function mentioned above is used to keep better solutions in the current generation. Two-point crossover is performed to produce the next generation, and mutations are carried out randomly according to a given probability. For detailed settings, please see the Evaluation section.

The evolution stops when the best solution in the current generation remains constant comparing with the previous generation, a situation indicating that the fitness value has reached convergence (*i.e.* the difference of fitness values between generations is less than a given convergence threshold). The abstract composite service binds concrete component services according to the best solution.

Note that it is not required to specially distinguish core services from non-core services in the application of GA since the weight value of any core component service is absolutely larger than non-core ones. Concrete services with better QoS for the core ser-

vice nodes are relatively easy to be found and kept using our weight-based fitness function. In other words, quality of core services can be guaranteed using the proposed approach.

3.4 New Fault Handling Mechanism: Partial Execution

Traditionally, composite services fail when any internal component services are missing or broken. Thus, fault handling actions are recommended to avoid a failure of the entire composite service. In accordance with this notion and to support our idea of core services, we devised a new fault handling mechanism, called partial execution (see Fig. 5). Prior to the execution of composite services, a list of service candidates is arranged to store concrete services corresponding to the core abstract services with a weight greater than a given threshold. During the execution, partial execution results are compiled while the service flow completes the invocation of a core service. When any failure occurs, the system attempts to overcome the problem using the following process: the system attempts to find a substitute service to instantly re-bind the service if the failed service is a core component. In cases in which the problem cannot be fixed by swapping services, the system returns the results from the available partial executions.

We used the same travel planning service as an example to illustrate the fault handling process shown in Fig. 5. The core services were "book airline tickets" and "reserve hotel rooms"; therefore, we applied partial execution mechanism for those two services (Fig. 6). According to the type of service failure, fault handling proceeds as follows:

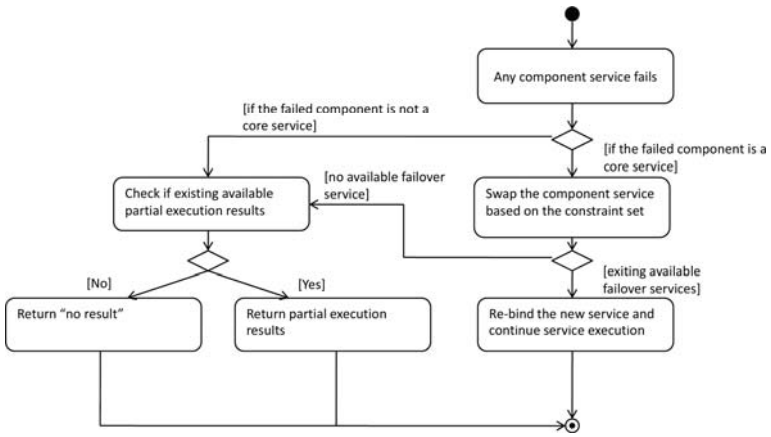

Fig. 5. Proposed fault handling mechanism: partial execution.

1. Failure of core services:
   – Case 1: Failure to "S_2 – book airline tickets": is a substitute candidate service available?
     • Yes: Swap a candidate service and continue execution.
     • No: Inform user that partial results are unavailable (the "book airline tickets" service is unavailable).
   – Case 2: Failure to "S_7 – reserve hotel rooms": is a substitute candidate service available?
     • Yes: Swap a candidate service and continue execution.

- No: Send user the partial execution results (up to "book airline tickets").

2. Failure of non-core services:
   – Case 1: Failure to "S_1 – inquiry of available airline tickets": inform user that no results are available.
   – Case 2: Failure to S_3, S_4, S_5 and S_6: send user the partial execution results (up to "book airline tickets" service).
   – Case 3: Failure to S_8, S_9, and S_{10}: send user the partial execution results (up to "reserve hotel rooms" service).

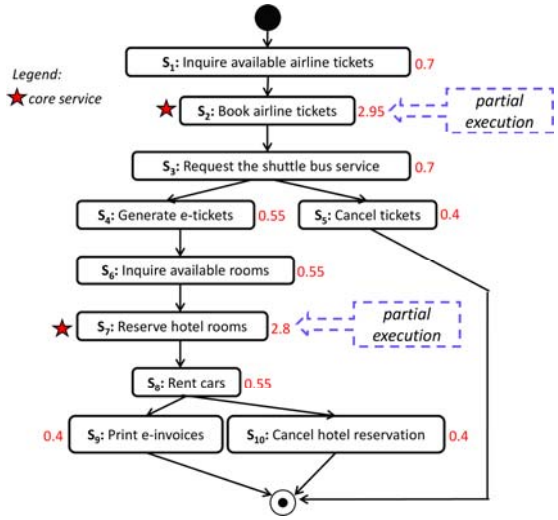

Fig. 6. Applying partial execution in the travel service.

In our blueprint, partial execution is a fault-tolerant method that can enhance the availability of composite services. When any component service in a composite service is interrupted, an attempt will be made to repair it using an available replacement service. We propose methods to deal with any situation and list the conditions we can control as follows:

1. If no core service is replaced during execution of a composite service, the composite service is considered healthy and the output process is returned to the requester.
2. In the event that a core concrete service fails and is fixed using a prepared candidate service, we still consider the process a success.
3. If the process proceeds through the first core service without being replaced, and the next non-core service fails, the partial execution results of the first core service are returned to the requester. As illustrated in Fig. 6, when the process crosses a core service (book airline tickets) and invokes it successfully, the output of the service is immediately retained. If the next non-core service (*i.e.* "generate e-tickets", "cancel tickets", or "inquire about available rooms") fails, the prior result "book airline tickets" will be returned to the requester as a partial execution result. The service requester can at least obtain significant results from the consumption of the composite service. The availability of the composite service is still maintained under this condition.

4. EVALUATION

This section describes the verification of the proposed approach and discusses possible threats to the proposed approach.

4.1 Evaluation Method

The devised experiments demonstrate the ability of the framework to ensure the smooth operation of core component services and the application of dissimilar flow structures. The experiment process was based on two subjects: (1) travel composite services (the motivating example) and (2) a virtual complex process including all structures (switch, parallel, and loop). The setup of each subject comprised abstract services corresponding to 100 concrete services, the 2.0 of the core service threshold, the 0.7 of I_u illustrated in Eq. (4), and the parameters of GA. The parameters of GA included a population of one hundred for each generation, Elitist GA (the best 40% of the individuals are kept alive over subsequent generations), the 0.8 probability of crossover, the 0.001 probability of mutation, and the 0.0000000001 of the convergence threshold. Finally, we recorded the data of the best chromosome (composite service) in every generation, repeated the whole procedures for 100 times, and calculated the average fitness values for the whole service flow and the core component services.

Our approach emphasizes the quality of service and core component services. We analyzed the differences between the fitness values of the composite service gained from Eq. (5) and the core services when applying weight analysis (our proposed approach) and not applying weight analysis (the widely-cited method proposed by Canfora *et al.* [11, 21]).

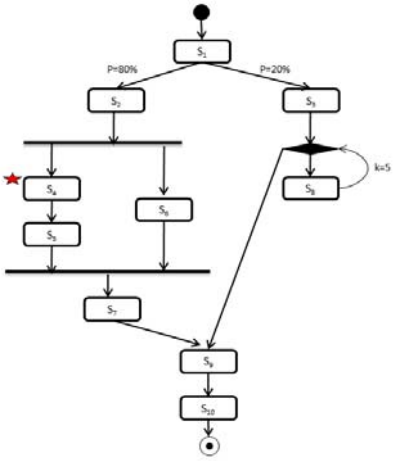

Fig. 7. Complex process (including switch, parallel and loop structures).

In the first experiment, the core services of the travel composite service were “book airline tickets” and “reserve hotel rooms” which were defined using Eq. (4) to obtain the final weights of 2.95 and 2.8, both of which exceeded the established threshold of 2.0.

We studied the quality of these core services according to the fitness value. In the second experiment, we applied our system to a structurally complex process including all structures (switch, parallel and loop), as shown in Fig. 7. The final weight for individual component services within a complex structure was obtained using Eq. (4). The weight of S_4 was 2.108, and the core service for the complex process was S_4. To evaluate our approach with the most complex situation, we set $S_4 - S_5$ and S_6 in a parallel configuration.

Table 2 shows the experiment results of these two experiments. For the first experiment, although the fitness values of the proposed approach for the travel service are somewhat lower than those of the original method that does not consider the notion of core services, the fitness value of the proposed approach for the first core service "book air airline tickets" and the second core service "reserve hotel rooms" are definitely higher than that of the original method. This means that the proposed approach is capable of obtaining the optimal composite service with highest possible quality core services. For the second experiment, the experiment results shown in Table 2 similar to the first one, indicating that our approach can be utilized for complex structures associated with QoS-aware selection. Notably, in the second experiment, the quality of core service using our approach is much better than the original method. Besides, to ensure the quality of the core services, our system is capable of issuing partial executions (mentioned in section 3.4) to ensure that, regardless of where an execution fails, complete or partial results of the process are returned to the customer.

Table 2. Experiment results.

	Average Fitness Value using Original GA	Average Fitness Value using Core-Service-Based GA	Comparison
Overall Travel Service	0.0356	0.0304	–14.60%
Core Service 1 of Travel Service	2.1618	2.3846	**10.31%**
Core Service 2 of Travel Service	2.0727	2.8261	**36.35%**
Overall Complex Service	0.1716	0.1513	–11.86%
Core Service of Complex Service	1.4048	2.5417	**80.93%**

4.2 Discussion

In this section, we discuss possible threats for the proposed approach. First, why not consider other methods to find service combinations, such as integer programming, is a possible question. Although the integer programming technique may solve most optimization problems, it cannot directly address the issue of complex control structures such as selection or parallel process. In [4, 17], Zeng *et al.* applied the integer programming method to select services of each execution path and merge multiple solutions as a final solution. However, the computational cost of this kind of approaches increases exponentially with the number of choice structures in the composition, and multiple execution paths may produce conflicting service selections [16]. Thus, we choose genetic algorithms to avoid above risks.

Second, it may be arguable whether the process designer (not the end user) has the ability to determine the importance scores of all component services. This issue can be

explained from two viewpoints: (1) the designer must be familiar with the goal of the devised service composition, and supposedly, the designer can identify what services are more important than others; (2) Some decision making mechanism, such as AHP (Analytic Hierarchy Process) [25], can be applied to assist the process designer in assigning appropriate points. The process of decision making is beyond the scope of this study.

5. CONCLUSIONS

This paper presents a novel approach to dynamically compose services based on weighted service flow and genetic algorithms. In the proposed approach, every abstract component service in the composition is assigned a weight value to represent its importance according to the definition set by the process designer and execution path analysis. The GA-based process yields a service combination including the selected concrete component services. The advantage of the proposed approach is the ability to search for the service composition while maintaining acceptable quality level and to ensure that the identified core component services (which are most likely to influence the overall service flow) are more robust than the others. Besides, using the proposed approach, the availability of composite services can be maintained since quality of core services is ensured and fault tolerance is achieved through the partial execution mechanism. Finally, experiments were conducted to evaluate effectiveness, and the results demonstrate the proposed approach can effectively maintain the quality of core component services.

In a future study, we plan to apply a tree-coded structure or AI-planning approach to achieve totally dynamic service composition, which means that both the structure and service binding will be performed dynamically.

REFERENCES

1. F. Curbera, R. Khalaf, N. Mukhi, *et al.*, "The next step in web services," *Communications of the ACM*, Vol. 46, 2003, pp. 29-34.
2. J. Lee, S.-P. Ma, Y.-Y. Lin, *et al.*, "Dynamic service composition: A discovery-based approach," *International Journal of Software Engineering and Knowledge Engineering*, Vol. 18, 2008, pp. 199-222.
3. N. Milanovic and M. Malek, "Current solutions for web service composition," *IEEE Internet Computing*, Vol. 8, 2004, pp. 51-59.
4. L. Zeng, B. Benatallah, A. H. H. Ngu, *et al.*, "QoS-aware middleware for web services composition," *IEEE Transations on Software Engineering*, Vol. 30, 2004, pp. 311-327.
5. R. Aggarwal, V. Kunal, J. Miller, and W. Milnor, "Constraint driven web service composition in METEOR-S," in *Proceedings of IEEE International Conference on Services Computing*, 2004, pp. 23-30.
6. L. Qi, Y. Tang, W. Dou, and J. Chen, "Combining local optimization and enumeration for QoS-aware web service composition," in *Proceedings of IEEE International Conference on Web Services*, 2010, pp. 34-41.
7. M. Alrifai, T. Risse, and W. Nejdl, "A hybrid approach for efficient web service composition with end-to-end QoS constraints," *ACM Transactions on Web*, Vol. 6,

2012, pp. 1-31.
8. Z. Wu, A. Ranabahu, K. Gomadam, *et al.*, "Automatic composition of semantic web services using process mediation," in *Proceedings of International Conference on Enterprise Information Systems*, 2007, pp. 453-461.
9. W. Jiang, C. Zhang, Z. Huang, *et al.*, "QSynth: A tool for QoS-aware automatic service composition," in *Proceedings of IEEE International Conference on Web Services*, 2010, pp. 42-49.
10. H. Zheng, J. Yang, and W. Zhao, "QoS analysis and service selection for composite services," in *Proceedings of IEEE International Conference on Services Computing*, 2010, pp. 122-129.
11. G. Canfora, M. D. Penta, R. Esposito, and M. Villani, "A framework for QoS-aware binding and re-binding of composite web services," *Journal of Systems and Software*, Vol. 81, 2008, pp. 1754-1769.
12. X. Ye and R. Mounla, "A hybrid approach to QoS-aware service composition," in *Proceedings of IEEE International Conference on Web Services*, 2008, pp. 62-69.
13. Y.-Y. Fanjiang, Y. Syu, C.-H. Wu, *et al.*, "Proceedings of the genetic algorithm for QoS-aware dynamic web services composition," in *Proceedings of International Conference on Machine Learning and Cybernetics*, 2010, pp. 3246-3251.
14. F. Rosenberg, M. B. Muller, P. Leitner, *et al.*, "Metaheuristic optimization of large-scale QoS-aware service compositions," in *Proceedings of IEEE International Conference on Services Computing*, 2010, pp. 97-104.
15. M. Matskin and J. Rao, "Value-added web services composition using automatic program synthesis," in *Web Services*, *E-Business*, *and the Semantic Web*, Vol. 2512, C. Bussler, *et al.*, eds., Springer, Berlin, Heidelberg, 2002, pp. 213-224.
16. C.-W. Lan, R. C. S. Chen, A. Y. S. Su, *et al.*, "A multiple objectives optimization approach for QoS-based web services compositions," in *Proceedings of IEEE International Conference on e-Business Engineering*, 2009, pp. 121-128.
17. L. Zeng, B. Benatallah, M. Dumas, *et al.*, "Quality driven web services composition," in *Proceedings of the 12th International Conference on World Wide Web*, 2003, pp. 411-421.
18. M. Alrifai and T. Risse, "Combining global optimization with local selection for efficient QoS-aware service composition," in *Proceedings of the 18th International Conference on World Wide Web*, 2009, pp. 881-890.
19. F. G. A. de Oliveira Jr. and J. M. P. de Oliveira, "QoS-based approach for dynamic web service composition," *Journal of Universal Computer Science*, Vol. 17, 2011, pp. 712-741.
20. T. Weise, S. Bleul, D. Comes, and K. Geihs, "Different approaches to semantic web service composition," in *Proceedings of the 3rd International Conference on Internet and Web Applications and Services*, 2008, pp. 90-96.
21. G. Canfora, M. D. Penta, R. Esposito, and M. Villani, "An approach for QoS-aware service composition based on genetic algorithms," in *Proceedings of Conference on Genetic and Evolutionary Computation*, 2005, pp. 1069-1075.
22. L. Aversano, M. D. Penta, and K. Taneja, "A genetic programming approach to support the design of service compositions," *International Journal of Computer Systems Science and Engineering*, Vol. 21, 2006, pp. 247-254.
23. C. Gao, M. Cai, and H. Chen, "QoS-aware service composition based on tree-coded

genetic algorithm," in *Proceedings of the 31st Annual International Computer Software and Applications Conference*, 2007, pp. 361-367.
24. M. Hutcheson, *Software Testing Fundamentals: Methods and Mtrics*, John Wiley & Sons, Inc., NY, 2003.
25. T. L. Saaty, *Fundamentals of Decision Making and Priority Theory with the Analytic Hierarchy Process*, RWS Publications, USA, 2000.

Shang-Pin Ma (馬尚彬) received his Ph.D. and BS degrees in Computer Science and Information Engineering from National Central University, Chungli, Taiwan, in 2007 and 1999, respectively. He has been an Assistant Professor of Computer Science and Engineering Department, National Taiwan Ocean University, Keelung, Taiwan, since 2008. His research interests include web-based software engineering, service-oriented computing, and semantic web.

Yong-Yi FanJiang (范姜永益) received the BS degree in Computer Science and Information Engineering from Tamkang University, Taiwan, in 1994. He received his MS and Ph.D. degree in Computer Science and Information Engineering from National Chiao Tung University, Taiwan, in 1998 and from National Central University, Taiwan, in 2004, respectively. Currently, he is an Assistant Professor in the Department of Computer Science and Information Engineering, Fu Jen Catholic University, Taiwan, from 2007. His research interests include mobile and pervasive computing, software engineering, semantic web, and human computer interaction.

Jong-Yih Kuo (郭忠義) received his BS degree from National Tsing Hua University, Taiwan, in 1991, and his Ph.D. degree from the National Central University, Taiwan, in 1998. He is now an Assistant Professor in the Intelligent System Laboratory of the Department of Computer Science and Information Engineering at the National Taipei University of Technology in Taiwan. His research interests include agent-based software engineering and fuzzy logic.