Abstract

Composite service evolution is one of the most important challenges to deal with in the field of service composition. In particular, this paper presents, LiveMig, an approach to live migration of composite service instance, which is a critical step for online composite service evolution. In LiveMig, a set of change operations preserving soundness is first defined. Second, a live instance state migration algorithm is proposed to determine if the state migration is allowed or not, and to compute the exact state after the migration. Finally, the correctness of LiveMig is theoretically proved, and an extensive set of simulations are performed to show its feasibility and effectiveness.

1. Introduction

Service composition is widely considered as an effective method to support the development of business applications using loosely-coupled and distributed web services over the Internet[1]. Currently, process-oriented (process-aware) service composition has become the research mainstream in services computing, which borrows the idea from traditional workflow technique and Business Process Management (BPM) methodology*.

Due to the highly dynamic changes of business requirements and Internet environments, composite service evolution is one of the most important challenges to deal with in the field of service composition. The dynamic evolution requires composite services to be capable of self-adapting according to the changes of business requirements and runtime environments. The main exhibition of dynamic composite service evolution is changeability of component services, adjustability and configurability of structural relations[2].

The realization of dynamic composite service evolution is faced with a lot of technical challenges[3-6]. In particular, there is an imperative need for an approach that supports the live instance migration when an old composite service definition is changed to a new one. When a composite service definition is experiencing dynamic evolution, its instances may still be running at the same time. For example, in order to satisfy new business requirements, a composite service for an e-commerce order-processing in a SCM (Supply Chain Management) system may need to partially adjust the business logic so as to support the new business pattern defined by RosettaNet PIP. Specifically, the original financial reporting function needs to be extended to support the external examination from a third-party auditing company. With this extension, the newly proposed Sarbanes-Oxley can be supported. In these scenarios, business services must run for 7*24 hours without any interruption, and the demand of dynamic evolution is more urgent. These critical scenarios neither allow composite service instances to be aborted and rolled back due to expensive time costs, nor allow the existence of multiple versions of composite service instances because the old versions are not compliant with new business constraints. Therefore we need an approach to live instance migration in composite service evolution.

For live instance migration in composite service evolution, early research[3, 7, 8] identifies a challenging problem “dynamic change bug”, which means the states of composite service instances of an old process definition does not have any corresponding states that exist in the new one. The change regions are adopted to restrict the migration of some instances that are under certain states (e.g. invalid structural adjustment). However, the computational complexity of change regions is proved to be very high[8], thus process inheritance[9] and relaxed process projection[10] methods are proposed. But these methods do not support structural adjustments of business processes and provide no solutions to computing the new state after evolution.

Another important issue is the correctness guarantee during composite service evolution. In process-oriented composite service, the most important correctness criterion is structural soundness (or soundness for short)[3, 11]. For a complex business process it may be intractable to verify the soundness. Especially under the scenarios of online composite service evolution, there is not enough time to do the verification. In a word, it is highly important and challenging to find an effective method that can satisfy evolution requirements while preserve soundness after composite service evolution.

*We shall henceforth not discriminate between the two terms of composite service and business process. In this paper, composite services also refer to process-oriented composite services.
In addition, in order to evaluate further actual effectiveness of LiveMig, we have made simulation experiments. A simulation program is developed with JDK1.5 and runs in a PC (Intel core 2 1.86G CPU, 1G memory). All the experiments are repeated 10 times, and we report the average results. Simulation experiment 1 explains the relationship between amount of composite service live instance and migration time (solid line shown in Figure 6a). In the experiment, old composite service definition containing 20 component services is a sound and is created at random using set of building blocks (Chapter 4.3.3 in book [11]). Based on the proposed set of change operations, a new definition of composite service is obtained after implementing 10 changes at random. We vary the number of concurrent live instances from 100 to 1000 to get various results respectively. Simulation experiment 2 shows the relationship between size of composite service and migration time (solid lines shown in Figure 6b). In the experiment, currently there are 1000 live instances in the old component service definition. A new composite service definition is obtained after implementing 10 changes at random. We vary the number of the component services in old composite service definition from 5 to 40 (created at random) to get various results respectively. Our LiveMig just aims at migration of one live instance. When a large number of instances is simultaneously migrated, the algorithm can be appropriately optimized in two ways. On the one hand the reachability graph of evolved composite service may be created one time; on the other hand a lot of live instances at same state is not necessary to be computed repeatedly. After optimization, we have renewedly made experiment to get dasheds in Figure 6a and Figure 6b. It can be seen that performance of instances migration has been obviously improved.

8. Conclusion

In this paper, we introduce LiveMig, an approach to live instance migration in composite service evolution. First, we give a set of change operations preserving soundness and prove that the soundness of new composite services after evolution can be satisfied after using change operations in the basic operation set, which helps to avoid complex verifying process. Second, a live instance state migration algorithm is proposed to determine if a state migration is allowed or not, and to compute the exact state after the migration. Finally, the correctness of LiveMig is theoretically proved, and an extensive set of simulations are performed to show its feasibility and effectiveness. Our future work includes QoS-aware evolution process control and multi-version management.

Acknowledgment: This work was supported by the National High Technology Research and Development Program of China (863 program) under grant 2007AA010301, 2006AA01A106 and 2009AA01Z419. We would like to thank Yipeng Ji for the experiments in this paper.

References