On Automated Composition for Web Services *

Zhongnan Shen and Jianwen Su
Department of Computer Science
University of California, Santa Barbara
szn@cs.ucsb.edu, su@cs.ucsb.edu

ABSTRACT

We develop a framework to compose services through discovery and orchestration for a given goal service. Tightening techniques are used in composition algorithms to achieve "completeness".

Categories and Subject Descriptors

H.1.m [**Models and Principles**]: Miscellaneous; D.2.8 [**Software Engineering**]: Metrics—*performance measures*

General Terms

Design, Algorithm, Measurement

Keywords

Service Composition, Service Discovery, Goal Service, Tightening, Completeness

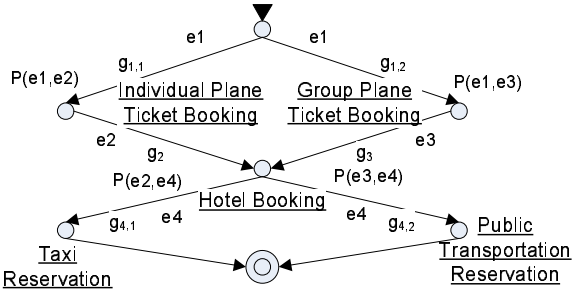
1. INTRODUCTION

Automated service composition, valuable in many domains, is among the promises enabled by SOA. Motivated by applications in e-commerce and e-science, we outline in this short note a framework for service composition that starts from a *stateful* goal service. Given the fact that existing and soon-to-be-existing services are mostly described by stateless WSDL standard, an immediate service composition problem is to map the activities in the goal service to existing (WSDL) services through querying service registries. Clearly, service discovery has to be integrated into service composition in the composition algorithms. We study issues concerning algorithms for the service composition problem.

2. SERVICE COMPOSITION

A *goal service* is specified as a stateful web service using automata, composed of activities that are (1) connected with combined data and control flows and (2) described with entry/exit conditions. Each *activity* in a goal service is described with input/output envelops (data flows) and entry/exit conditions (semantics). An *envelop* is a set of attribute names with associated types. Attributes in envelops may be referenced in conditions. Given an instance of an input envelop, the activity can be performed if the *entry condition* is satisfied; after the activity, an instance of an output envelop is generated and the *exit condition* for this pair of input and output envelops holds. Once the goal service is specified, relevant services can be discovered through searching service registries.

Fig. 1 shows a *travel agent* goal service. The service gets a travel request from the user, books plane tickets, and makes transportation reservations. Depending on the arrival time of the flight, the travel agent makes a taxi reservation or a public transportation reservation. Due to the space limitation, we only describe the path with *individual plane ticket booking* and *public transportation reservation*. The goal service guarantees day time arrival, which is expressed as "$\texttt{EstArrTime} \in [6, 20]$" in the entry condition $g_{1,1}$. The entry condition $g_{4,2}$ of the *public transportation reservation* says that tickets are reserved if the arrival time is between 6am and 10pm ($\texttt{ArrTime} \in [6, 22]$). $P(e_1, e_2)$ is the exit condition of the *individual plane ticket booking* activity, which says the arrival time is within the estimated time range.



$e_1 = [\texttt{DepPort}, \texttt{ArrPort}, \texttt{ArrDate}, \texttt{NumPeople}, \texttt{EstArrTime}, \texttt{EstPlanePrice}, \texttt{EstHotelPrice}, \texttt{RoomType}]$;
$e_2 = [\texttt{ArrPort}, \texttt{ArrDate}, \texttt{NumPeople}, \texttt{ArrTime}, \texttt{PlanePrice}, \texttt{EstHotelPrice}, \texttt{RoomType}]$;
$e_4 = [\texttt{ArrPort}, \texttt{ArrDate}, \texttt{NumPeople}, \texttt{ArrTime}, \texttt{HotelPrice}, \texttt{HotelName}]$;
$g_{1,1}: \ e_1.\texttt{NumPeople} \leqslant 5 \wedge e_1.\texttt{EstArrTime} \subseteq [6, 20]$;
$g_{4,2}: \ e_4.\texttt{ArrTime} \in [6, 22]$;
$P(e_1, e_2) = e_2.\texttt{ArrTime} \in e_1.\texttt{EstArrTime} \wedge \cdots$.

Figure 1: A travel agent goal service

The composition problem is to identify for each activity in a goal service a "suitable" service in a registry. Services in repository are stateless services described by input/output envelops and pre-/post-conditions. Such a composition framework is motivated by scientific workflow [4] and interactive services [1, 3]. We assume that the control flow of a goal service is acyclic.

A *realization* for a goal service over a registry is a mapping from activities in the goal service to services in the registry. For each activity with input envelops $e^{\text{i}}_1, ..., e^{\text{i}}_n$, output envelops $e^{\text{o}}_1, ..., e^{\text{o}}_m$, entry conditions g_j for e^{i}_j, and exit conditions $P(e^{\text{i}}_j, e^{\text{o}}_k)$ ($1 \leqslant j \leqslant n, 1 \leqslant k \leqslant m$), the service assigned to the activity has input envelop e^{i}, output envelop e^{o}, precondition P, and post-condition Q such that

*Supported in part by NSF grants IIS-0415195 and CNS-0613998.

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

- $e^{\mathrm{i}} \subseteq \cap_{1 \leqslant j \leqslant n} e^{\mathrm{i}}_j$, $e^{\mathrm{o}} \supseteq \cup_{j,k}(e^{\mathrm{o}}_j - e^{\mathrm{i}}_k)$,
- $\vee_{1 \leqslant j \leqslant n} g_j \rightarrow P$, $Q \rightarrow \wedge_{j,k} P(e^{\mathrm{i}}_j, e^{\mathrm{o}}_k)$.

The input envelop and the precondition of the assigned service should be guaranteed by the goal service, while the output envelop and the exit condition of the goal service should be covered by the assigned service. Instead of finding services using non-functional properties [6], service discovery here is based on conditions [2]. The service composition problems are stated as follows.

Schema composition problem: Given a goal service G and a service registry R, find a feasible realization of G over R.

Instance composition problem: Given a goal service G, a service registry R, and a goal invocation V, find a feasible realization of G over R for the invocation V.

3. COMPOSITION ALGORITHMS

A naive composition algorithm to compute a realization is to formulate a search $(\cap_j e^{\mathrm{i}}_j, \cup_{j,k}(e^{\mathrm{o}}_j - e^{\mathrm{i}}_k), \vee_j g_j, \wedge_{j,k} P(e^{\mathrm{i}}_j, e^{\mathrm{o}}_k))$ for each activity against service registries. However, this may miss candidate services. For example, services which provide public transportation less than 6am to 10pm are not qualified for the *public transportation reservation* activity in Fig. 1.

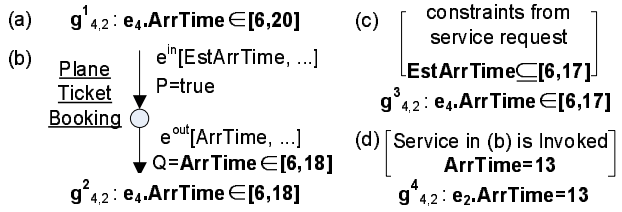


Figure 2: Tightening the goal service

Indeed, the entry condition and the exit condition of the *individual plane ticket booking* activity can be used together to tighten $g_{4,2}$ to $\texttt{ArrTime} \in [6, 20]$, as shown in Fig. 2(a). Also, if the service in Fig. 2(b) is selected for the activity, the entry condition $g_{4,2}$ can be further tightened using the post-condition of the selected service. For instance composition problem, constraints from the service request (or goal invocation) can be pushed down through the goal service and help tighten the entry conditions, as shown in Fig. 2(c). Finally, with a goal invocation, selected services can be invoked. As an example, if the service in Fig. 2(b) is invoked, attributes in the output envelop are bound to specific values, which can be used to tighten $g_{4,2}$ to $g^4_{4,2}$ (Fig. 2(d)).

Conditions of preceding activities, selected services, and constraints in service requests can all be used to tighten entry conditions of the succeeding activities. Tighter entry conditions increase the possibility of finding qualified services. For example, after the entry condition is tightened as discussed above, public transportation services become qualified if they serve before 8pm instead of 10pm. We developed two strategies for tightening a goal service.

Tighten Strategy I aims at the schema composition problem. Entry conditions and exit conditions of preceding activities are pushed down to tighten entry conditions of the following activities. Once an activity has been assigned a service in the registry, the exit conditions are replaced by the post-condition of the selected service.

Tighten Strategy II aims at the instance composition problem. Constraints from the goal invocation as well as conditions from preceding activities are pushed down through the goal service. Once an activity is assigned a service in registry, the service selected is invoked, and attributes in output envelop are bound to specific values. These conditions are added to the exit conditions of the activity.

A composition algorithm with a tightening strategy works as follows. If it is an instance composition, constraints from a goal invocation are attached to the goal service. From the first activity to the last (goal services are acyclic), the algorithm generates the tightened entry conditions using conditions from preceding activities, and replaces the original entry conditions with the tightened ones. Subsequently, individual service discovery is performed for the activity. A service is nondeterministically selected from the search result. Finally, the exit conditions of the activity are updated by being replaced with the the post-condition of the selected service. The selected service is invoked and value-binding conditions are added to the exit conditions if it is an instance composition. See [5] for the complexity of the tightening process.

4. COMPOSITION COMPLETENESS

Tightening strategies naturally lead to the question whether a composition algorithm is "complete". By completeness, we mean the ability to find a feasible realization if it exists. There are two notions of completeness for the two composition problems. A composition algorithm is *schema complete* if for every service registry and any goal service, the algorithm computes a realization whenever it exists. A composition algorithm is *instance complete* if for each service registry, each goal service, and each goal invocation, the algorithm returns a realization whenever the goal service has a realization with services in the registry for the invocation.

For schema completeness, the only information about services is from service descriptions. The tightening strategy I makes a full use of the service descriptions, and a schema composition algorithm with the first tightening strategy is shown to be schema complete. For instance completeness, if the tightening strategy II is deployed, the composition algorithm can be shown to be instance complete. However, to achieve instance completeness, it needs to do exhaustive search and invoke all the related services.

5. CONCLUSIONS

Automated web service composition using service discovery is an interesting practical problem in many application domains. Algorithms are developed for composing a given stateful goal service through discovering stateless services in registries. Completeness of composition algorithms are investigated. There are many interesting problems worth further exploration. One direction is the development of efficient and yet effective algorithms that can utilize properties of specific data domains and condition formats. Incremental or online algorithms are also very interesting. Another direction is to re-examine query functions a registry should support.

6. REFERENCES

[1] D. Berardi, D. Calvanese, G. De Giacomo, R. Hull, and M. Mecella. Automatic composition of transition-based semantic web services with messaging. *VLDB*, 2005.

[2] I. Elgedawy, Z. Tari, and M. Winikoff. Exact functional context matching for web services. *ICSOC*, 2004.

[3] C. Gerede, R. Hull, O. Ibarra, and J. Su. Automated composition of e-services: Lookaheads. *ICSOC*, 2004.

[4] B. Ludaescher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific workflow management and the kepler system. *Journal of Concurrency and Computation*, To appear.

[5] Z. Shen and J. Su. On complexity of the tightening problem for web service discovery. *SOCA 2007*, To appear.

[6] L. Zeng, B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. Qos-aware middleware for web services composition. *IEEE Trans. on Soft. Engineering*, 30(5):311–327, 2004.