Towards a General Framework for Web Service Composition

Srividya Kona, Ajay Bansal, M. Brian Blake
Department of Computer Science,
Georgetown University
Washington, DC 20057

Gopal Gupta
Department of Computer Science,
The University of Texas at Dallas
Richardson, TX 75083

Abstract

Service-oriented computing (SOC) has emerged as the eminent market environment for sharing and reusing service-centric capabilities. The underpinning for an organization's use of SOC techniques is the ability to discover and compose Web services. In this paper we present a generalized semantics-based technique for automatic service composition that combines the rigor of process-oriented composition with the descriptiveness of semantics. Our generalized approach extends the common practice of linearly linked services by introducing the use of a conditional directed acyclic graph (DAG) where complex interactions, containing control flow, information flow and pre/post conditions, are effectively represented.

1. Introduction

Service-oriented computing is changing the way software applications are being designed, developed, delivered, and consumed. A composite service is a collection of services combined together in some way to achieve a desired effect. Traditionally, the task of automatic service composition has been split into four phases: (i) Planning, (ii) Discovery, (iii) Selection, and (iv) Execution [2]. Most efforts reported in the literature focus on one or more of these four phases. The first phase involves generating a plan, i.e., all the services and the order in which they are to be composed in order to obtain the composition. The plan may be generated manually, semi-automatically, or automatically. The second phase involves discovering services as per the plan. Depending on the approach, often planning and discovery are combined into one step. After all the appropriate services are discovered, the selection phase involves selecting the optimal solution from the available potential solutions based on non-functional properties like QoS properties. The last phase involves executing the services as per the plan and in case any of them are not available, an alternate solution has to be used.

In this paper we formalize the generalized composition problem based on our conditional directed acyclic graph representation. We present our approach that generates most general compositions based on (conditional) directed acyclic graphs (DAG). In our framework, the DAG representation of the composite service is reified as an OWL-S description. This description document can be registered in a repository and is thus available for future searches. The composite service can now be discovered as a direct match instead of having to look through the entire repository and build the composition solution again.

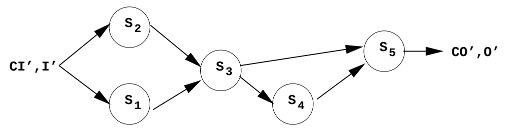

Figure 1. Example of a Composite Service as a Directed Acyclic Graph

2. Web service Composition

In this section we formalize the generalized composition problem. In this generalization, we extend our previous notion of composition [1] to handle non-sequential conditional composition (which we believe is the most general case of composition). Informally, the Web service Composition problem can be defined as follows: given a repository of service descriptions, and a query with the requirements of the requested service, in case a matching service is not found, the composition problem involves automatically finding a directed acyclic graph of services that can be composed to obtain the desired service. Figure 1 shows an example composite service made up of five services $\mathcal{S}_1$ to $\mathcal{S}_5$. In the figure, I' and CI' are the query input parameters and pre-conditions respectively. O' and CO' are the query output parameters and post-conditions respectively. Informally, the directed arc between nodes $\mathcal{S}_i$ and $\mathcal{S}_j$ indicates that outputs of $\mathcal{S}_i$ constitute (some of) the inputs of $\mathcal{S}_j$.

Definition (Repository of Services): Repository ($\mathcal{R}$) is a set of Web services.

Definition (Service): A service is a 6-tuple of its pre-conditions, inputs, side-effect, affected object, outputs and post-conditions. $\mathcal{S} = (\mathcal{CI}, \mathcal{I}, \mathcal{A}, \mathcal{AO}, \mathcal{O}, \mathcal{CO})$ is the representation of a service where $\mathcal{CI}$ is the list of pre-conditions, $\mathcal{I}$ is the input list, $\mathcal{A}$ is the service's side-effect, $\mathcal{AO}$ is the affected object, $\mathcal{O}$ is the output list, and $\mathcal{CO}$ is the list of post-conditions. The pre- and post-conditions are ground logical predicates.

Definition (Query): The *query service* is defined as $\mathcal{Q}$ = $(\mathcal{CI}', \mathcal{I}', \mathcal{A}', \mathcal{AO}', \mathcal{O}', \mathcal{CO}')$ where $\mathcal{CI}'$ is the list of pre-conditions, $\mathcal{I}'$ is the input list, $\mathcal{A}'$ is the service affect, $\mathcal{AO}'$ is the affected object, $\mathcal{O}'$ is the output list, and $\mathcal{CO}'$ is the list of post-conditions. These are all the parameters of the requested service.

Definition (Generalized Composition): The generalized Composition problem can be defined as automatically finding a directed acyclic graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ of services from repository $\mathcal{R}$, given query $\mathcal{Q} = (\mathcal{CI}', \mathcal{I}', \mathcal{A}', \mathcal{AO}', \mathcal{O}', \mathcal{CO}')$, where $\mathcal{V}$ is the set of vertices and $\mathcal{E}$ is the set of edges of the graph. Each vertex in the graph either represents a service involved in the composition or post-condition of the immediate predecessor service in the graph, whose outcome can be determined only after the execution of the service. Each outgoing edge of a node (service) represents the outputs and post-conditions produced by the service. Each incoming edge of a node represents the inputs and pre-conditions of the service. The following conditions should hold on the nodes of the graph:

1. $\forall i\ \mathcal{S}_i \in \mathcal{V}$ where $\mathcal{S}_i$ has exactly one incoming edge that represents the query inputs and pre-conditions, $\mathcal{I}' \sqsupseteq \bigcup_i \mathcal{I}_i$, $\mathcal{CI}' \Rightarrow \wedge_i \mathcal{CI}_i$.
2. $\forall i\ \mathcal{S}_i \in \mathcal{V}$ where $\mathcal{S}_i$ has exactly one outgoing edge that represents the query outputs and post-conditions, $\mathcal{O}' \sqsubseteq \bigcup_i \mathcal{O}_i$, $\mathcal{CO}' \Leftarrow \wedge_i \mathcal{CO}_i$.
3. $\forall i\ \mathcal{S}_i \in \mathcal{V}$ where $\mathcal{S}_i$ represents a service and has at least one incoming edge, let $\mathcal{S}_{i1}$, $\mathcal{S}_{i2}$, ..., $\mathcal{S}_{im}$ be the nodes such that there is a directed edge from each of these nodes to $\mathcal{S}_i$. Then $I_i \sqsubseteq \bigcup_k \mathcal{O}_{ik} \cup I'$, $CI_i \Leftarrow (\mathcal{CO}_{i1} \wedge \mathcal{CO}_{i2} \ldots \wedge \mathcal{CO}_{im} \wedge CI')$.
4. $\forall i\ \mathcal{S}_i \in \mathcal{V}$ where $\mathcal{S}_i$ represents a condition that is evaluated at run-time and has exactly one incoming edge, let $\mathcal{S}_j$ be its immediate predecessor node such that there is a directed edge from $\mathcal{S}_j$ to $\mathcal{S}_i$. Then the inputs and pre-conditions at node $\mathcal{S}_i$ are $I_i = \mathcal{O}_j \cup I'$, $CI_i = \mathcal{CO}_j$. The outgoing edges from $\mathcal{S}_i$ represent the outputs that are same as the inputs I_i and the post-conditions that are the result of the condition evaluation at run-time.

The meaning of the $\sqsubseteq$ is the subsumption (subsumes) relation and $\Rightarrow$ is the implication relation. In other words, a service at any stage in the composition can potentially have as its inputs all the outputs from its predecessors as well as the query inputs. The services in the first stage of composition can only use the query inputs. The union of the outputs produced by the services in the last stage of composition should contain all the outputs that the query requires to be produced. Also the post-conditions of services at any stage in composition should imply the pre-conditions of services in the next stage. When it cannot be determined at compile time whether the post-conditions imply the pre-conditions or not, a *conditional* node is created in the graph. The outgoing edges of the conditional node represent the possible conditions which will be evaluated at run-time. Depending on the condition that holds, the corresponding services are executed. That is, if a subservice $\mathcal{S}_1$ is composed with subservice $\mathcal{S}_2$, then the postconditions $\mathcal{CO}_1$ of $\mathcal{S}_1$ must imply the preconditions $\mathcal{CI}_2$ of $\mathcal{S}_2$. The following conditions are evaluated at run-time:
if $(\mathcal{CO}_1 \Rightarrow \mathcal{CI}_2)$ *then execute* $\mathcal{S}_1$;
else if $(\mathcal{CO}_1 \Rightarrow \neg\, \mathcal{CI}_2)$ *then no-op*;
else if $(\mathcal{CI}_2)$ *then execute* $\mathcal{S}_1$;

3. Automatic Generation of Composite Services

In order to produce the composite service which is the graph, we filter out services that are not useful for the composition at multiple stages. The composition routine starts with the query input parameters. It finds all those services from the repository which require a subset of the query input parameters. For the next stage, the inputs available are the query input parameters and all the outputs produced by the previous stage, i.e., $\mathcal{I}_2 = \mathcal{O}_1 \cup I$. $\mathcal{I}_2$ is used to find services at the next stage, i.e., all those services that require a subset of $\mathcal{I}_2$. In order to make sure we do not end up in cycles, we get only those services which require at least one parameter from the outputs produced in the previous stage. This filtering continues until all the query output parameters are produced. At this point we make another pass in the reverse direction to remove redundant services which do not directly or indirectly contribute to the query output parameters. This is done starting with the output parameters working our way backwards.

4. Conclusions and Future Work

To make Web services more practical we need a general framework for composition of Web services. The generalized approach presented in this paper can handle non-sequential conditional composition that can be used in automatic workflow generation in a number of applications.

References

[1] S. Kona, A. Bansal, and G. Gupta. Automatic Composition of Semantic Web Services. In *ICWS, 2007.*

[2] J. Cardoso, A. Sheth. Semantic Web Services, Processes and Applications. *Springer, 2006.*