

Large-Scale Service Composition in Semantic Service Discovery

Steffen Bleul, Thomas Weise and Kurt Geihs
Distributed Systems Group
University of Kassel, Germany
{bleul, weise, geihs}@vs.uni-kassel.de

Abstract

Self-Healing and self-optimizing service based applications are important steps towards the self-organizing Service Oriented Architectures (SOA). Self-Organizing SOAs replace services by functional equivalent services in the case of faults or in respect of quality of service. These features depend on automatic service discovery which provides service alternatives. We enter the WSC'06 contest to present a semantic service discovery system for large sets of services. A recursive algorithm builds service compositions by adding services in each iteration. The search works backwards, since we add services that produce a certain output regardless of its input parameters. A valid service composition produces a set of queried output parameters and input parameters necessary for the composed services. The algorithm is improved by using efficient data structures in our service composition system.

1. Introduction

A Service Oriented Architecture provides an arbitrary number of services. Services can be seen as software components. Therefore, a SOA can provide building blocks for software development. Applications built on services can be self-healing and self-optimizing, as the services used can be changed during runtime with respect to the service availability and the quality of the services.

The success of such processes depends on the service discovery. Service discovery searches for a set of services that provides the given functionality. Manual service discovery is cumbersome in large sets of services. The task is even more complicated if we also want to consider the compositions of services that provide the functionality.

Semantic service discovery includes the annotation of input and output parameters with semantic concepts. Semantic concepts reflect the semantic meaning of parameters and semantic relationships to each other. Such relationships are expressed by taxonomies which express a hierarchical rela-

tionship between concepts [5], comparable to inheritance in object-oriented programming.

We present a system which addresses the WSC'06 challenge on automatic semantic service discovery and service composition [2]. We introduce a search algorithm and performant data structures for semantic deduction and service matching for service discovery and service composition.

This paper is structured as follows: In section 2 we introduce our semantic composition system. Our approach is presented in section 2.1 and the task of these components for the challenge of both semantic service discovery and semantic service composition is presented in section 2.2. Afterwards we elaborate our thoughts on optimizations in section 2.3. Section 3 is about the future direction of our work and in section 4 we conclude the paper.

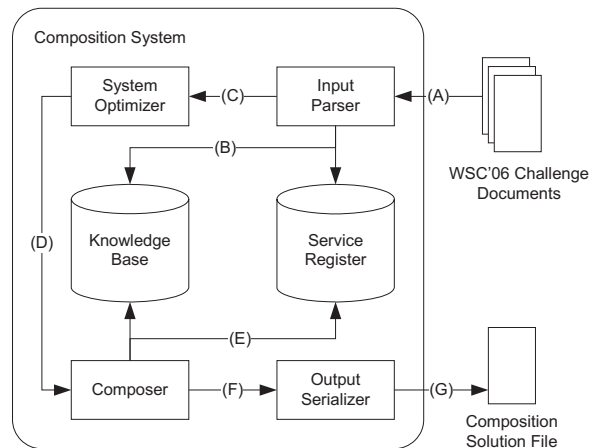


Figure 1. Semantic Composition System

2. Technical Description

Our composition system consists of six components as presented in figure 1. The *Input Parser* is configured to

Preview

This document is a preview version and not necessarily identical with the original.

parse WSC'06 challenge documents (A). The WSDL [7] service interface description documents provided are put into the *Service Register* (SR) and the XSD file which contains the taxonomy of semantic concepts is used to fill the *Knowledge Base* (KB) (B). Afterwards the *Input Parser* provides information, e.g. the size of the SR and the KB, to the *System Optimizer* (C). The *System Optimizer* decides the necessary optimization techniques (see section 2.3) for the composer.

The composer is used both for the challenge of semantic service discovery and the semantic service composition. Our system handles service discovery as a composition with the maximum size of one service. The *Composer* works on the data provided by the KB and SR (E). Each found composition is passed to the *Output Serializer* (F) which serializes our data structures in an WSC'06 challenge conformant solution file (G).

2.1. Semantic Service Discovery

The WSC'06 challenge provides service interface descriptions where input and output parameters are described by semantic concepts. The first task is to discover all services that produce a set of output concepts for a set of input-concepts and thus produce the queried output parameters. The second task is to discover all compositions of services that produce the queried output parameters.

We have developed a semantic composition algorithm that solves both the tasks of service discovery and composition of services. For the first task we limit the composition size to one service. For the second task the algorithm builds a service composition backwards. We do not look for services that can be started with the input concepts; rather we search for services that produce the desired output concepts. Our search is successful if our produced output concepts match all input concepts of all composed services and the queried output concepts.

Matching of the semantic concepts is done by querying the KB for the predicate $subsumes(A, B)$ for concept A and B . The predicate is evaluated as true when A is an equivalent or a subsumption of concept B . In this case the parameter annotated with concept B is an equivalent or a specialization of the parameter with concept A . Thus concept B represents a parameter that is the same as the parameter annotated with concept A or an inheritance of the parameter annotated with concept A . Therefore, we consider concept B as a match of concept A because the parameter includes the necessary information.

2.2. Service Composition Algorithm

The algorithm uses recursion to build paths of services which represent compositions. The algorithm is defined as follows:

```

1 Input :
2
3  $\alpha$ : The composition currently investigated.
4  $\beta$ : The set of demanded output concepts.
5  $\gamma$ : The set of input concepts already known.
6
7 Algorithm discoverComposition( $\alpha, \beta, \gamma$ ) {
8
9   for each  $s$  in promisingServices( $\beta, \gamma$ ) {
10      $\beta_{new} = updateSearchedParameters(\beta, s, \gamma)$ ;
11
12     if ( $\beta_{new} = \emptyset$ ) {
13       Output( $\alpha \cup s$ );
14       updateCancelCriteria();
15       return;
16     }
17
18     if ( $|\alpha| \geq maxPath$ )
19       return;
20
21     if testSearchCancel()
22       return;
23
24      $\gamma_{new} = updateKnownParameters(\beta_{new}, \alpha, \gamma)$ ;
25
26     discoverComposition( $\alpha \cup s, \beta_{new}, \gamma_{new}$ );
27   }
28
29 Algorithm call:
30
31 discoverComposition( $\emptyset, \beta_{query}, \gamma_{query}$ );
32
33  $\beta_{query}$ : The set of queried output concepts.
34  $\gamma_{query}$ : The set of queried input concepts.

```

It is started with the query input concepts γ_{query} and output concepts β_{query} . First we build a set of services that we could append to the path in the next recursion step (line 9). The function $promisingServices(\beta, \gamma)$ therefore delivers a priority queue of services. Each service in this queue delivers at least one output concept which is currently unknown in our path. The priority queue is arranged in a descending order with respect to the number of additional output concepts the service produces for our concept list γ . We improve our discovery by first following paths with services that produce more outputs.

Afterwards we produce a list β_{new} of missing concepts needed to cover all input concepts of services already added to our path by calling $updateSearchedParameters(\beta, s, \gamma)$. If the list β_{new} is empty (line 12) then we have found a valid composition and $Output(\alpha \cup s)$ first adds the current service to our path which is then passed to the *OutputSerializer* (line 13). Otherwise we call $updateKnownParameters(\beta_{new}, \alpha, \gamma)$. This function

updates the list γ_{new} of known input concepts γ by adding the outputs of those services already part of path α which have no unsatisfied input parameters. Afterwards we add the current service to our path and start another recursion step with the new build path (line 26).

Additionally we have search termination and path elimination criteria. The condition in line 18 limits the search depth which is necessary for service discovery which are compositions with recursion depth one. In line 14 we mark the last time we have found a solution with *updateCancelCriteria()*. The function *testSearchCancel()* in line 21 represents further path elimination criteria dependent on memory usage, used computation time, the time we last found a solution and the number of paths currently followed.

2.3. System Optimization

The service register (SR) consists of layered hashes. The first layer hash will be instantiated for each request and it contains volatile request specific data, while the second level hash contains the provided immutable WSDL database. These hashes contain objects which represent a service. Each service object (SO) stores a set of its input parameters and a set of output parameters. The hash keys are the output parameters semantic concepts and the data items are lists of services that provide that output concept. Thus, references to a SO will be stored multiple times if it produces multiple outputs. We model service compositions as SOs. If a valid composition has been created, its sub-compositions are stored in the first level hash, effectively merging paths and allowing a fast reuse.

For fast deduction of the predicate *subsumes(A, B)* we use a hash-tree-combination as Knowledge Base (KB). In a hash, we store all the nodes of a tree, where the nodes have links to their parent nodes. Each node represents one concept. This way the KB is able to deduct if one semantic concept *A* subsumes another concept *B* by finding an entry point in the tree for concept *B* and traversing the tree backwards to find concept *A*. After the node is found, the search is done in time $O(d)$ where d is the depth of the tree.

Two other hashes are used to improve the performance by saving successfully evaluated subsume-searches in one hash and unsuccessful ones in the other. We traverse the tree if we find no entry in both hashes. This speeds up the evaluation of the predicate *subsumes(A, B)*.

Further data structures improve our search algorithm. Their use is dependent on the available memory at runtime and the number of services and concepts. Therefore, we use the *System Optimizer* component in our system to decide for a given number of services and concepts which data structures and optimization strategies are used at runtime.

3. Future Work

The introduced composition system will be part of our automatic service brokering system in the research project ADDO [1]. Although the Input Parser and Output Serializer are implemented especially for the WSC'06 contest, it is easily reuseable in our context. The Input Parser will be replaced by a parser for semantic interface descriptions like OWL-S [3] or WSMO [6]. Thus, our system can perform service discovery with existing Semantic Web Service technologies.

The output serializer will be adjusted to transform valid WSC'06 compositions to OWL-S processes or BPEL4WS [4] descriptions. Therefore, the results are executable service orchestrations which can be used as replacements for faulting or insufficient services in service based applications.

4. Conclusion

We have presented a composition system for large sets of services. The system solves the WSC'06 challenge on semantic service discovery and service composition. The introduced algorithm produces service compositions and is currently improved by optimizing the necessary data structures. We are also developing further path elimination techniques to enhance its performance further.

References

- [1] Automatic service brokering in service oriented architectures.
URL: <http://www.vs.uni-kassel.de/research/addo/>.
- [2] Web service challenge 2006.
URL: <http://www.ws-challenge.org/>.
- [3] D. M. et al. *OWL-S, OWL-based Web Service Ontology*, 2004.
URL: <http://www.daml.org/services/owl-s/1.1/>.
- [4] IBM. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, 2003.
- [5] M. C. Jaeger, G. Rojec-Goldmann, G. M. hl, C. Liebethuth, and K. Geihs. Ranked Matching for Service Descriptions using OWL-S. In P. Müller, R. Gotzhein, and J. B. Schmitt, editors, *Kommunikation in verteilten Systemen (KiVS 2005)*, Informatik Aktuell, pages 91–102, Kaiserslautern, Germany, February 2005. Springer Press.
- [6] D. Roman, U. Keller, and H. Lausen. Wsmo - web service modeling ontology. In *DERI Working Draft 14*. Digital Enterprise Research Institute (DERI), February 2004.
- [7] World Wide Web Consortium. *WSDL, Web Services Description Language*, August 2005.
URL: <http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803/>.

```

@conference{BWG2006WSC,
author      = {Steffen Bleul and Thomas Weise and Kurt Geihs},
title       = {Large-Scale Service Composition in Semantic Service Discovery},
booktitle   = {Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology
               (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)},
editor      = {Ws-Challenge Part: M. Brian Blake and Andreas Wombacher and Michel C.
               Jaeger and William K. Cheung},
ISBN        = {ISBN-10: 0-7965-2511-3, ISBN-13: 978-0-7695-2511-2},
pages       = {427--429},
publisher   = {IEEE Computer Society, Los Alamitos, California, Washington, Tokyo},
year        = {2006},
month       = jun,
affiliation = {University of Kassel},
location    = {The Westin San Francisco Airport, 1 Old Bayshore Highway, Millbrae,
               United States},
note        = {We won the Web Service Challenge on Semantic Service Discovery and
               Composition 2006, see Web Service Challenge Home, Conference Home,
               Results Page (Access to the software is restricted.), order number
               P2511, Library of Congress Number 2006927609\
               The work is online available at
               http://www.it-weise.de/documents/index.html\#BWG2006WSC.\
               The publication can be downloaded at
               http://www.it-weise.de/documents/files/BWG2006WSC.pdf.\
               The presentation can be downloaded at
               http://www.it-weise.de/documents/files/BWG2006WSC\_slides.pdf.\
               The software can be downloaded at
               http://www.it-weise.de/documents/files/BWG2006WSC\_software.zip.\
               Contact Thomas Weise at tweise@gmx.de or http://www.it-weise.de/},
abstract     = {Self-Healing and self-optimizing service based applications are
               important steps towards the self-organizing ServiceOriented
               Architectures (SOA). Self-Organizing SOAs replace services by functional
               equivalent services in the case of faults or in respect of quality of
               service. These features depend on automatic service discovery which
               provides service alternatives. We enter the WSC'06 contest to present a
               semantic service discovery system for large sets of services. A
               recursive algorithm builds service compositions by adding services in
               each iteration. The search works backwards, since we add services that
               produce a certain output regardless of its input parameters. A valid
               service composition produces a set of queried output parameters and
               input parameters necessary for the composed services. The algorithm is
               improved by using efficient data structures in our service composition
               system.},
contents     = {* Introduction\
               * Technical Description\
               - Semantic Service Discovery\
               - Service Composition Algorithm\
               - System Optimization\
               * Future Work\
               * Conclusion},
keywords     = {Web Service Composition, Web Service Discovery, Semantic Web, Semantic
               Web Service Composition, Web Service Challenge},
language     = {en},
url          = {http://csdl2.computer.org/persagen/DLabsToc.jsp?resourcePath=/dl/proceedings/\&toc=comp/proceedings/cec-eee/2006/2511/00/2511toc.xml \&DOI=10.1109/CEC-EEE.2006.59}
}

```