

# Semantic Web Service Composition for Service-Oriented Architectures

Thomas Weise, Steffen Bleul, Marc Kirchhoff, Kurt Geihs  
University of Kassel  
Wilhelmshöher Allee 73  
34121 Kassel  
Email: {weise|bleul|kirchhoff|geihs}@vs.uni-kassel.de

## Abstract

*Semantic web service composition is about finding services from a repository that are able to accomplish a specified task. The task is defined in a form of a composition request which contains a set of available input parameters and a set of wanted output parameters. Instead of the parameter values, concepts from an ontology describing their semantics are passed to the composition engine. The composer works on a repository of services. The parameters of these services are semantically annotated in the same way as the parameters in the request. The composer then finds a set of services fulfilling the request – the composition. If the input parameters given in the request are provided, the services of this set can be executed and will finally produce the wanted output parameters. In this paper, we introduce our new, improved composition system with which we will take part in the Web Service Challenge 2008.*

## Preview

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

<http://www.it-weise.de/>

## 1. Introduction

The necessity for fast service composition systems is directly connected with the emergence of Service-Oriented Architectures (SOAs). Today, the software of an enterprise must be able to adapt to changes in the business processes like accounting, billing, the workflows, and even in the office software. Thus, corporate software has to be built with the anticipation of changes and updates [1]. A SOA is the

ideal architecture for such systems [2, 3]. Service-Oriented architectures allow us to modularize the business logic and to implement it in the form of services which are accessible in a network. Services are building blocks for service processes which represent the workflows of an enterprise. They can be added, removed, and updated at runtime without interfering with the ongoing business. A SOA can be seen as a complex system with manifold services as well as  $n:m$  dependencies between services and applications:

- An application may need various service functionalities.
- Different applications may need the same service functionality.
- A certain functionality may be provided by multiple services.

Self-organizing approaches for managing these dependencies often rely on a combination of syntactic and semantic service descriptions in order to decide whether a service provides a functionality being sought or not. Common syntactic definitions like WSDL specify the order and types of service parameters and return values. Semantic interface description languages like OWL-S [4] or WSMO [5] annotate these parameters with a meaning. While WSDL can be used to define a parameter `myisbn` of the type `String`, with OWL-S we can define that `myisbn` expects a `String` which actually contains an `ISBN`. Via a taxonomy we can now deduce that values which are annotated as either `ISBN-10` or `ISBN-13`<sup>1</sup> can be passed to this service.

A wanted functionality is defined by a set of required output and available input parameters. A service offers this functionality if it can be executed with the given input parameters and its return values contain the needed output values. Many service management approaches employ semantic service discovery [6, 7, 8]. Still, there is a substantial

<sup>1</sup>There are two formats for International Standard Book Numbers (ISBNs), ISBN-10 and ISBN-13, see also <http://en.wikipedia.org/wiki/ISBN> [accessed 2007-09-02].

lack of research on algorithms and system design for fast response service discovery. This is especially the case in service composition where service functionality is not necessarily provided by a single service. Instead, combinations of services, the so-called *compositions*, are discovered.

In this paper we not only want to illustrate our approach for the WS-Challenge 2008 but also want to point out our ADDOaction semantic service discovery and service integration framework. The framework already applies semantic service discovery for service replacement and uses semantic matchmaking for appropriate mediation between the clients and the services interchanging SOAP messages. After introducing semantic service composition in addition to discovery of single services, the challenge results can provide salient advantages inside SOAs.

## 2. Semantic Service Composition

Let us define some prerequisites in order to discuss the idea of semantic service composition properly. Therefore, let us assume that all concepts in the knowledge base are members of the set  $\mathbb{M}$  and can be represented as nodes in a wood of taxonomy trees.

**Definition 1 (subsumes)** *Two concepts  $A, B \in \mathbb{M}$  can be related in one of four possible ways. We define the predicate  $subsumes : (\mathbb{M} \times \mathbb{M}) \mapsto \{true, false\}$  to express this relation as follows:*

1.  $subsumes(A, B)$  holds if and only if  $A$  is a generalization of  $B$  ( $B$  is then a specialization of  $A$ ).
2.  $subsumes(B, A)$  holds if and only if  $A$  is a specialization of  $B$  ( $B$  is then a generalization of  $A$ ).
3. If neither  $subsumes(A, B)$  nor  $subsumes(B, A)$  holds,  $A$  and  $B$  are not related to each other.
4.  $subsumes(A, B)$  and  $subsumes(B, A)$  is  $true$  if and only if  $A = B$ .

*The subsumes relation is transitive, and so are generalization and specialization.*

If a parameter  $x$  of a service is annotated with  $A$  and a value  $y$  annotated with  $B$  is available, we can set  $x = y$  and call the service only if  $subsumes(A, B)$  holds (*contravariance*). This means that  $x$  expects less or equal information than given in  $y$ .

From the viewpoint of a composition algorithm, there is no need for a distinction between parameters and the annotated concepts. The set  $\mathbb{S}$  contains all the services  $s$  known to the registry. Each service  $s \in \mathbb{S}$  has a set of required input concepts  $s.in \subseteq \mathbb{M}$  and a set of output concepts  $s.out \subseteq \mathbb{M}$  which it will deliver on return. We can trigger a service if we can provide all of its input parameters.

Similarly, a composition request  $R$  always consists of a set of available input concepts  $R.in \subseteq \mathbb{M}$  and a set of requested output concepts  $R.out \subseteq \mathbb{M}$ . A composition algorithm discovers a (topologically sorted) set of  $n$  services  $S = \{s_1, s_2, \dots, s_n\} : s_1, \dots, s_n \in \mathbb{S}$ . As shown in Equation 1, the first service ( $s_0$ ) of a valid composition can be executed with instances of the input concepts  $R.in$ . Together with  $R.in$ , its outputs ( $s_1.out$ ) are available for executing the next service ( $s_2$ ) in  $S$ , and so on. The composition provides outputs that are either annotated with exactly the requested concepts  $R.out$  or with more specific ones (*covariance*). For each composition solving the request  $R$ ,  $isGoal(S)$  will hold:

$$\begin{aligned} isGoal(S) \Leftrightarrow & \forall A \in s_1.in \exists B \in R.in : subsumes(A, B) \wedge \\ & \forall A \in s_i.in, i \in \{2..n\} \\ & \exists B \in R.in \cup s_{i-1}.out \cup \dots \cup s_1.out : subsumes(A, B) \wedge \\ & \forall A \in R.out \exists B \in s_1.out \cup \dots \cup s_n.out \cup R.in : \\ & subsumes(A, B) \end{aligned} \quad (1)$$

The search space that needs to be investigated in web service composition [?] is the set of all possible permutations of all possible sets of services. In order to proceed in this space, we define the operation *promising* which obtains the set of all services  $s \in \mathbb{S}$  that produce an output parameter annotated with the concept  $A$  (regardless of their inputs).

$$\forall s \in promising(A) \exists B \in s.out : subsumes(A, B) \quad (2)$$

## 3. Heuristics – Fast and Efficient

In our past experiences, heuristic searches are the best performing approaches for semantic web service composition. Here, a heuristic  $c$  helps to decide which nodes are to be expanded next. As main composition method we therefore define a greedy algorithm that internally sorts the list of currently known candidate compositions in *descending* order according to a heuristic in form of a comparator function  $c : \mathbb{S}^n \in \mathbb{R}$ . The comparator function  $c(S_1, S_2)$  will be below zero if  $S_1$  seems to be closer to the solution than  $S_2$  and greater than zero if  $S_2$  is a more prospective candidate. Thus, the best elements will be at the end of the list  $X$  in Algorithm 1.

We can easily derive different comparator functions for web service composition. In our experiments and real-world experiences, the function  $c_{cmp}$  has proven to be most efficient. It combines the size of the set unsatisfied parameters  $\forall A \in wanted(S) \Rightarrow \exists s \in S : A \in s.in \wedge A \notin known(S)$ , the composition lengths, the number of satisfied parameters  $\forall B \in eliminated(S) \Rightarrow \exists s \in S : B \in s.in \wedge B \in known(S)$ , and the number of known concepts  $known(S) = R.in \cup_{\forall s \in S} s.out$  as defined in Algorithm 2.

---

**Algorithm 1:**  $S = greedyComposition(R, c)$ 

---

**Input:**  $R$  the composition request  
**Data:**  $X$  the sorted list of compositions to explore  
**Output:**  $S$  the solution composition found, or  $\emptyset$

```
1 begin
2    $X \leftarrow \bigcup_{A \in R.out} (promising(A))$ 
3   while  $X \neq \emptyset$  do
4      $X \leftarrow sort(descending, X, c)$ 
5      $S \leftarrow popLastElement(X)$ 
6     if  $isGoal(S)$  then return  $S$ 
7     foreach  $A \in wanted(S)$  do
8       foreach  $s \in promising(A)$  do
9          $X \leftarrow appendList(X, s \oplus S)$ 
10  return  $\emptyset$ 
11 end
```

---

---

**Algorithm 2:**  $r = c_{cmp}(S_1, S_2)$ 

---

**Input:**  $S_1, S_2$  two composition candidates  
**Output:**  $r \in \mathbb{Z}$  indicating whether  $S_1$  ( $r < 0$ ) or  $S_2$  ( $r > 0$ ) should be expanded next

```
1 begin
2    $i_1 \leftarrow |wanted(S_1)|$ 
3    $i_2 \leftarrow |wanted(S_2)|$ 
4   if  $i_1 = 0$  then
5     if  $i_2 = 0$  then return  $|S_1| - |S_2|$ 
6     else return  $-1$ 
7   if  $i_2 = 0$  then return  $1$ 
8    $e_1 \leftarrow |eliminated(S_1)|$ 
9    $e_2 \leftarrow |eliminated(S_2)|$ 
10  if  $e_1 > e_2$  then return  $1$ 
11  else if  $e_1 < e_2$  then return  $-1$ 
12  if  $i_1 > i_2$  then return  $1$ 
13  else if  $i_1 < i_2$  then return  $-1$ 
14  if  $|S_1| \neq |S_2|$  then return  $|S_1| - |S_2|$ 
15  return  $|known(S_1)| - |known(S_2)|$ 
16 end
```

---

First, it compares the number of wanted parameters. If a composition has no such unsatisfied concepts, it is a valid solution. If both,  $S_1$  and  $S_2$  are valid, the solution involving fewer services wins. If only one of them is complete, it also wins. Otherwise, both candidates still have unsatisfied concepts. Only if both of them have the same number of satisfied parameters, we again compare the wanted concepts. If their numbers are also equal, we prefer the shorter composition candidate. If even the compositions are of the same length, we finally base the decision of the total number of known concepts.

## 4. Composition System Overview

An application accesses the composition system illustrated in Figure 1 by submitting a service request through its *Web Service Interface* and also provides the service descriptions and their semantic annotations in form of WSDL and XSD formatted files or OWL-S descriptions. These documents are parsed by a fast *SAX-based Input Parser*. The composition process itself is started by the *Strategy Planer* which allocates the system resources and chooses the composition algorithm that seems to be most appropriate for the given problem.

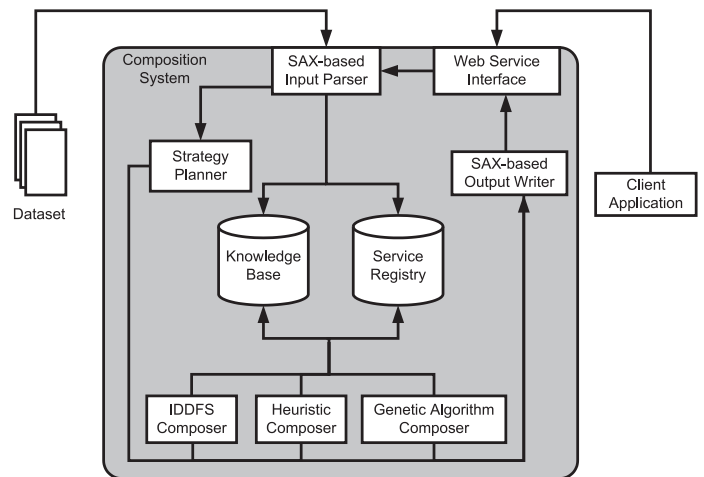


Figure 1: The composition system.

The software modules encapsulating the three composition algorithms have direct access to the *Knowledge Base* and to the *Service Register*. Although every algorithm and composition strategy is unique, they all work on the same data structures. One or more composition algorithms solve the composition challenge and pass the solution to a *SAX-based Output Writer*, a very fast XML output generator. The resulting document is then returned through the *Web Service Interface*.

The knowledge base and the service register are consti-

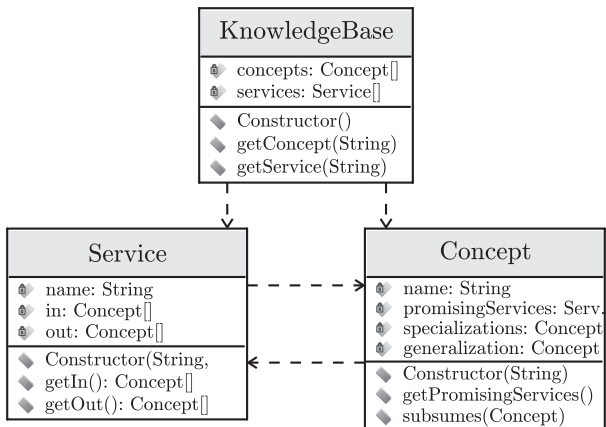


Figure 2: The knowledge base.

tuted by instances of the classes `Concept` and `Service`, as illustrated in Figure 2. Instances of `Concept` provides the method `promising` which corresponds to the predicate *promising* introduced in Equation 2. In order to determine its result, all the specializations of the concept have to be traversed and their promising services have to be accumulated. The crux of the routine is that this costly traversal is only performed once per concept. Our experiments have shown that the resource *memory* is not a bottleneck even for largest service repositories. Hence, `promising` caches its results. This caching is done in a way that is thread-safe on one hand and does not need any synchronization on the other. The same holds for all features of the knowledge base: multiple algorithms may run in parallel, using the same knowledge base without any synchronization-induced delay, race conditions or inconsistencies.

## 5. The ADDO System

Besides the heuristic method discussed in this paper, the ADDOaction<sup>2</sup> project additionally utilizes an iterative-deepening depth first search (IDDFS) approach and a Genetic Algorithm [9] for dynamic composition of software components.

The ADDOaction system is a semantic service discovery and integration framework for Web Services. Whereas the web service challenge applies service discovery based on WSDL documents and BPEL processes it does not yet define a sufficient model and adaption in terms of Web Services being exchanged at runtime. A fully automated approach requires further exploitation of syntactic and semantic aspects inside WSDL, an appropriate matching algorithm, and a comprehensive distributed architecture. In

this respect, the framework illustrates to which extend semantic service discovery can be used for automatic service mediation and, as a direct result, for service integration.

In [?] we presented the ADDOaction system and in [?], we demonstrate its application to BPEL orchestrations. The system uses WSDL descriptions enriched with the Mediation Contract Extension (MECE). It is able to generate the XSLT documents necessary to convert the arbitrarily complex message formats used for the data exchange between the services automatically. The most prominent features of the ADDO approach are the matching of XSD type definitions which include parameter units and data types, the specification of dependencies between required and optional service parameters, and sorting of message parameters. ADDO is not only able to match these features but also mediates between different identifiers, namespaces, message structures using XSLT and plug-ins for unit conversion. Each ADDO system component is available, monitorable, and manageable by Web Service interfaces and offers a high scalability. Business applications relying on ADDOaction as backbone can thus easily employ semantic service discovery for ad-hoc integration of business logic.

## 6. Conclusions

In this technical description, we have formally defined the problem of semantic web service composition. Based on this definition, we have derived a highly efficient composition algorithm based on an informed search approach utilizing a heuristic function. The ADDOaction software system has been described, which employs this algorithm along with two other approaches, an IDDFS search and a Genetic Algorithm. In practical experiments based on real world data and the data sets of the previous Web Service Challenges, this combination has proven to be very fast and reliable. We are eager to further improve this composition engine in the months to come and look forward to an exciting Web Service Composition 2008.

## References

- [1] Jabir and J. W. Moore from the 7803 Whiterim Terrace, Potomac, MD 20854, USA. A search for fundamental principles of software engineering. *Computer Standards & Interfaces*, 19(2):155–160, March 1998. doi:10.1016/S0920-5489(98)00009-9. Online available at [http://dx.doi.org/10.1016/S0920-5489\(98\)00009-9](http://dx.doi.org/10.1016/S0920-5489(98)00009-9) and <http://www.gelog.etsmtl.ca/publications/pdf/249.pdf> [accessed 2007-09-02].
- [2] Eric A. Marks and Michael Bell. *Executive's Guide to Service oriented architecture (SOA): A Planning and*

<sup>2</sup><http://www.vs.uni-kassel.de/ADDO/>

*Implementation Guide for Business and Technology*. John Wiley & Sons, Inc., Hoboken, NJ, April 2006. ISBN: 978-0-47003-614-3.

- [3] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, August 2, 2005. ISBN: 978-0-13185-858-9.
- [4] Anupriya Ankolekar, Mark Burstein, Grit Denker, Daniel Elenius, Jerry Hobbs, Lalana Kagal, Ora Lassila, Drew McDermott, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Marta Sabou, Craig Schlenoff, Evren Sirin, Monika Solanki, Naveen Srinivasan, Katia Sycara, and Randy Washington. *OWL-S 1.1 Release, OWL-based Web Service Ontology*. Web-Ontology Working Group at the World Wide Web Consortium, 2004. Online available at <http://www.daml.org/services/owl-s/1.1/> [accessed 2007-09-02].
- [5] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Ruben Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web Service Modeling Ontology. *Applied Ontology*, 1(1):77–106, 2005. See also <http://www.wsmo.org/> [accessed 2007-09-02] and [10].
- [6] Steffen Bleul and Thomas Weise from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. An Ontology for Quality-Aware Service Discovery. In C. Zirpins, G. Ortiz, W. Lamerdorf, and W. Emmerich, editors, *Engineering Service Compositions: First International Workshop, WESC05*, volume RC23821 of *IBM Research Report*. Yorktown Heights: IBM Research Division, December 12, 2005, Vrije Universiteit Amsterdam, The Netherlands. See also [7]. Online available at <http://www.it-weise.de/documents/files/BW2005QASD.pdf> [accessed 2008-09-04].
- [7] Steffen Bleul, Thomas Weise, and Kurt Geihs from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. An Ontology for Quality-Aware Service Discovery. *Computer Systems Science Engineering*, 21(4)(Special issue on “Engineering Design and Composition of Service-Oriented Applications”), July 2006. See also [6]. Online available at <http://www.it-weise.de/documents/files/BWG2006QASD.pdf> [accessed 2008-09-04].
- [8] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. A Framework for Designing and Composing Semantic Web Services. In *Semantic Web Services, First International Semantic Web Services Symposium, Proceedings of 2004 AAAI Spring Symposium Series*, March 22–24, 2004, History Corner, main quad (Building 200), Stanford University, CA, USA. Online available at <http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf> [accessed 2007-09-02]. See also [11].
- [9] Thomas Weise from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. *Global Optimization Algorithms – Theory and Application*. Thomas Weise, second edition, 2008. Online available at <http://www.it-weise.de/> [accessed 2008-09-04].
- [10] Dumitru Roman, Uwe Keller, and Holger Lausen. *WSMO – Web Service Modeling Ontology*. Digital Enterprise Research Institute (DERI), February 2004. Online available at <http://www.wsmo.org/2004/d2/v0.1/20040214/> [accessed 2007-09-02]. See also <http://www.wsmo.org/> [accessed 2007-09-02] and [5].
- [11] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. ODE SWS: A Framework for Designing and Composing Semantic Web Services. *IEEE Intelligent Systems*, 19(4):24–31, July-August 2004. ISSN: 1541-1672. doi:10.1109/MIS.2004.32. Online available at <http://iswc2004.semanticweb.org/demos/14/paper.pdf> [accessed 2007-09-02]. See also [8].

```

@inproceedings{WBKG2008SWSCFSOA,
  title      = {Semantic Web Service Composition for Service-Oriented
                Architectures},
  author     = {Thomas Weise and Steffen Bleul and Marc Kirchhoff and
                Kurt Geihs},
  booktitle  = {Proceedings of IEEE Joint Conference (CEC/EEE 2008) on E-
                Commerce Technology (Tenth CEC'07) and Enterprise Computing,
                E-Commerce and E-Services (Fifth EEE'08)},
  pages     = {355--358},
  year      = {2008},
  doi       = {10.1109/CEC/EEE/2008.69},
  booktitle  = {Proceedings of IEEE Joint Conference (CEC/EEE 2008) on
                E-Commerce Technology (Tenth CEC'07) and Enterprise
                Computing, E-Commerce and E-Services (Fifth EEE'08)},
  ISBN      = {978-0-7695-3340-7},
  publisher  = {IEEE Computer Society},
  month     = {jul # {~21--24}},
  location  = {Washington, D.C., District of Columbia, USA},
  institution = {Institute of Electrical and Electronics Engineers, Inc.
                (IEEE)},
  organization = {IEEE Computer Society},
  address    = {10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos,
                CA 90720-1314},
  copyright = {IEEE 2008},
  note      = {BMS Part Number CFP08321-PRT,
                Library of Congress Number 2005928254},
  url       = {http://www.it-weise.de/documents/files/WBKG2008SWSCFSOA.pdf}
}

```