

## Making a Fast Semantic Service Composition System Faster

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

### Abstract

*Service Oriented Architecture (SOA) is a flexible software design paradigm for enterprises. The workflows of a company are implemented as services which can be arranged, updated and managed at runtime without interfering with ongoing business. Service management aims at providing undisturbed access to services. Its efficiency strongly depends on a fast response time in the case of a failure. This is hard to achieve since the relations between applications and services require comprehensive knowledge and lack transparency for administrators. Self-organizing approaches promise a solution by automating service discovery. In this paper we present a service discovery system which enables service compositions from semantic descriptions stored in a knowledge base. Therefore, it utilizes multiple composition algorithms from which the most appropriate set is selected and applied according to the size of the knowledge base and the available processors. The functionality of our system is made available through a Web Service interface itself. It is thus applicable in self-organizing service management systems with any number of services and ontologies.*

### 1. Introduction

Today, companies must rely on an IT-architecture which is as flexible as its business strategy. Changes in the workflows, the vendors, the accounting and the documents of the enterprise directly lead to changes in its software. Hence, the software architecture has to anticipate changes and updates. A Service Oriented Architecture (SOA) allows these demands to be fulfilled without interfering with the ongoing business. In a SOA, business logic is implemented in the form of services accessible in a network. Services are building blocks for the workflows and resources of an enterprise and need to be manageable at runtime.

A SOA can be seen as a complex system with manifold service functionalities as well as dependencies between ser-

vices and application. Various applications are in need of certain service functionalities, which, in turn, may be provided by multiple services. Manual service management becomes more and more cumbersome and ineffective with a rising count of such relations. Self-organization promises a solution for finding services that offer a specific functionality automatically.

Such self-organizing approaches usually apply semantic service discovery. Here, services are specified by semantic interface description languages like OWL-S [4] in addition to syntactic ones, e.g. WSDL [11]. A particular functionality is defined by a set of required input and produced output parameters. A service offers this functionality if it can be executed with these provided input parameters and returns the wanted output parameters. The semantic concepts the parameters are annotated with are matched rather than their syntactic data types.

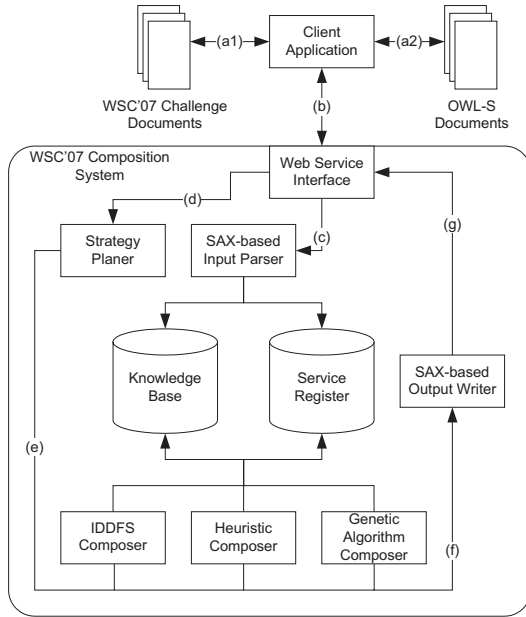
Many service management approaches employ semantic service discovery [1, 6, 7, 5]. Still there is a substantial lack of research on algorithms and system design for fast response service discovery. This is especially the case in service composition where service functionality is not necessarily provided by a single service. Instead, combinations of services are discovered.

The WSC [2] competition is the ideal platform to compare different approaches in this area as its core issue is not only research but a competition among scientists on fast semantic service composition. The positive response to our work in WSC'06 encouraged us to take measures to improve our successful and solid approach introduced in [3]. This extended research lead to enhancements to our 2006 algorithm which will be presented in this paper as part of the WSC'07.

The paper is structured as follows. In Section 2 we give an overview of our semantic composition system. We outline the system and give a detailed description of the algorithms and strategies in Section 3. The most important implementation issues are presented in Section 4. Section 5 discusses related work and projects. The paper closes with a conclusion in Section 6.

## 2. Architectural Design

We illustrate a more refined version of our last year's semantic composition system [3] in Figure 1.



**Figure 1. System Overview**

In order to provide the functionality of the composition algorithms to other software components, it was made accessible as a Web Service shortly after WSC'06. Hence, the web service composer is available for any system where semantic service discovery with the Ontology Web Language for Services (OWL-S) is used.

An application accesses the composition system by submitting a service request (illustrated by (b)) through its *Web Service Interface*. It furthermore provides the service descriptions and their semantic annotations. Therefore, WSDL and XSD formatted files, as used in the WSC challenge, or OWL-S descriptions have to be passed in ((a1) and (a2)). These documents are parsed by a fast *SAX-based Input Parser* (c). The composition process itself is started by the *Strategy Planer* (d). The Strategy Planer chooses an appropriate composition algorithm and instructs it with the composition challenge document (e).

The software modules containing the basic algorithms all 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 which are described in Section 4. One or more composition algorithm modules solve the composition challenge and pass the solution to a *SAX-based Output Writer*, an XML document generating module (f) faster than DOM serial-

ization. The result is afterwards returned through the *Web Service Interface* (g).

## 3. Semantic Service Composition

The composition system consists of three types of composition algorithms. The first one, an iterative deepening depth-first search (*IDDFS*) algorithm, is maintained in our system since it was the award winning solution of the WSC'06. It is especially fast in finding solutions for small and medium size service repositories. Additionally, we have extended our system with a heuristic and a genetic composition algorithm. The heuristic approach is well-suited for larger amounts of Web Services. If the service compositions also tend to be very large and complex, the genetic algorithm approach excels in speed clearly. From these three, the most appropriate one is selected by a strategy planner for each composition request.

All the algorithms search for an ordered set of services, which, if executed, produces the requested outputs. A composition request always consists of a set of available input parameters and a set of requested output parameters. We can trigger a service if we can provide its input parameters. After its completion, the service will return a set of output parameters as defined in its interface description. Composition algorithms work on service descriptions rather than executing services themselves. A composition algorithm recursively discovers a set of services that can be executed with the accumulated input parameters. Output parameters produced by these services are treated as available input parameters in the next iteration step. In semantic service composition, parameters are described by concepts rather than simple types. The compatibility of in and output parameters is thus specified by the relation between concepts within taxonomies.

The concept taxonomies define a hierarchy of semantic concepts. A concept  $A$  can either be the generalization or a specialization of a concept  $B$  or not related to it at all. These dependencies are specified by the predicate  $subsumes(A, B)$ . If this predicate evaluates to true,  $A$  is a generalization of concept  $B$  and  $B$  is a specialization of concept  $A$ . We can provide an input parameter annotated with concept  $A$  if the algorithm has accumulated a parameter with concept  $B$  and  $subsumes(A, B)$  is valid. If this is the case, we have either an input parameter of the required semantic type or a more specialized one which will carry at least the same information as its generalization. For this reason we also have found a requested output parameter annotated with concept  $A$  if the algorithm has accumulated a parameter with concept  $B$  and  $subsumes(A, B)$  is true.

We define the operation  $getPromisingServices$  which obtains the set  $S$  of services  $s \in S$  producing an output parameter annotated with the concept  $A$  in equation 1.

$$\forall s \in \text{getPromisingServices}(A) \exists B \in s.out : \text{subsumes}(A, B) \quad (1)$$

Based on this definition, we specify the simplest composition algorithm of our framework, the iterative deepening depth-first search, or short *IDDFS*, in algorithm 1. For

---

**Algorithm 1:** *IDDFS*(*in*, *out*)

---

**Input:** *in* the set of known input concepts  
**Input:** *out* the set of wanted output concepts  
**Data:** *maxDepth* the maximum search depth currently allowed  
**Data:** *set true* if a solution was found  
**Data:** *depth* the current depth  
**Data:** *composition* the current composition  
**Output:** a set of service compositions

*maxDepth*  $\leftarrow$  2  
*set*  $\leftarrow$  *false*  
**repeat**  
  | *internalSearch*(*in*, *out*,  $\emptyset$ , 1)  
  | *maxDepth*  $\leftarrow$  *maxDepth* + 1  
**until** *set*

**internalSearch**(*in*, *out*, *composition*, *depth*)  
**foreach** *o*  $\in$  *out* **do**  
  | *promising*  $\leftarrow$  *getPromising*(*o*)  
  | **foreach** *s*  $\in$  *promising* **do**  
    | *wanted*  $\leftarrow$  *out*  
    | **foreach**  $\omega \in s.out$  **do**  
      | *wanted*  $\leftarrow$  *wanted* \ { $\gamma : \text{subsumes}(\gamma, \omega)$ }  
    | **foreach** *i*  $\in$  *s.in* **do**  
      | **if**  $\nexists \gamma : \text{subsumes}(i, \gamma) \wedge \gamma \in \text{known}$   
      | **then** *wanted*  $\leftarrow$  *wanted*  $\cup$  *i*  
    | *composition*  $\leftarrow$  *s*  $\oplus$  *composition*  
    | **if**  $|\text{wanted}| \leq 0$  **then**  
      |  $\ll$  *solution found*  $\gg$   
      | *maxDepth*  $\leftarrow$  *depth*  
      | *set*  $\leftarrow$  *true*  
    | **else**  
      | **if** *depth* < *maxDepth* **then**  
        | *enhancedBruteForce*(*in*,  
        | *wanted*, *composition*,  
        | *depth* + 1)

---

each concept required but not yet known, the *IDDFS* algorithm checks all the promising services. By adding such a service *s* to a composition, all the generalizations of all its output parameters are no longer *wanted*. On the other hand, the input parameters of *s* which are not yet known become *wanted*.

An heuristic approach (called *hIDDFS*) builds up on this method by adding a heuristic function  $h(s)$  (see equation 2) which returns the ratio of the count of *wanted* output concepts that can be satisfied by adding the service to the current composition and the resulting composition length.

$$h(s) = \frac{\text{wanted}(\text{composition}) - \text{wanted}(\text{composition} \cup s)}{|\text{composition}| + 1} \quad (2)$$

The higher the value of *h*, the more likely is the service added to the composition. Here we also break with the *IDDFS* by allowing paths to grow in a range of  $1..n$  (instead strictly incrementing *maxDepth* as done in algorithm 1). If *n* is set to  $\infty$ , we have a straight heuristic search. If multiple processors are available, the same number of *hIDDFS* instances may run in parallel with different values of *n*.

Last but not least, the genetic algorithm grows a service composition by minimizing the three objective functions  $o_1$  to  $o_3$  in equations 3 to 5. It can be viewed as a generalization of many parallel heuristic searches, where the step of adding a service to a composition is no longer deterministic since it is performed by a randomized mutation operation.

$$o_1(\text{composition}) = |\text{wanted}(\text{composition})| \quad (3)$$

$$o_2(\text{composition}) = |\text{composition}| \quad (4)$$

$$o_3(\text{composition}) = \frac{1}{|\text{known}(\text{composition})| + 1} \quad (5)$$

$o_1$  favors compositions where only few required output parameters are still missing and  $o_2$  puts pressure in the direction of shorter compositions. The third objective function promotes compositions with services that provide lots of parameters as outputs. Adding new services to such compositions may produce fewer new “unknown” parameters.

Although pareto-optimization is the paradigm most often applied in multi-objective evolutionary computation, we decide for a hierarchical evaluation. Therefore, if a composition dominates another one in  $o_1$ , it will be chosen. If and only if both have equal  $o_1$ -values, the  $o_2$  values will be taken into consideration, and so on. In our tests this resulted in a faster convergence to the correct results.

## 4. Implementation

One of the most important implementation details is the realization of the operation *getPromisingServices* since it is used by all composition algorithms in each iteration step.

Each instance of *Concept* holds a list of services that directly produce a parameter annotated with it as output. The method *getPromisingServices*(*A*) of *Concept*, illustrated in Figure 2, additionally returns all the *Services* that provide a specialization of the concept *A* as output. In order to determine this set, 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 substantiated that the resource *memory*, even for largest service repositories, is not a bottleneck. Hence, *getPromisingServices* 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.

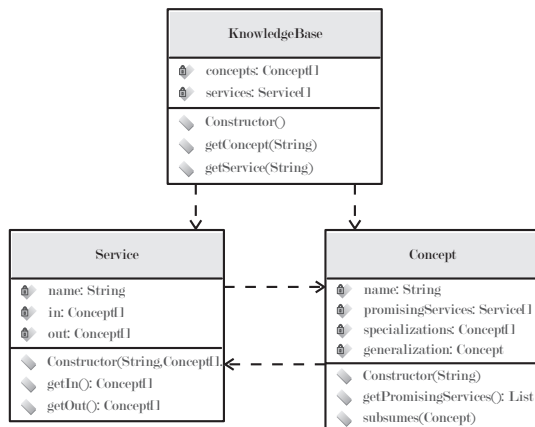


Figure 2. The knowledge base

The composition system is implemented using Java and is accessible through a Web Service interface.

## 5. Related Work

The ADDO [1] project uses the composition system presented in this paper as part of its architecture. The ADDO model distinguishes between configurable service containers as service proxies and a Service Broker. The Service Broker employs the composition system for service discovery and configures the service containers with newly discovered services for service replacement. Therefore, the system achieves self-healing properties.

The Distributed Genetic Programming Framework [9] provides the core of the genetic service composition algorithm. Originally it is a genetic programming environment [8] for the development of distributed algorithms such as proactive aggregation protocols [10]. For the context of the WSC we removed some components like logging, the graphical user interface, and the island-model distribution functionality for grids. The efficient evolutionary algorithms provided by the remaining core were customized for service composition.

## 6. Conclusion

This paper presents our semantic composition system which will enter the WSC'07. After winning the WSC'06, we discovered that our prior system could be further improved in order to deliver fast problem solving for any size of service repositories and semantic hierarchies. This versatility is achieved by utilizing three composition algorithms,

IDDFS, a heuristic, and a genetic composition algorithm, which complement each other.

Initial benchmarks indicate that the new composer reaches a speedup of up to 35% compared to the WSC'06 version.

Furthermore, we have equipped the composition system with a Web Service interface and included several import and export document formats like XSD, OWL, and OWL-S. The resulting system is applicable for all service management approaches that use semantic service discovery. As an integral part of the ADDO framework, it has also undergone severe testing.

## References

- [1] ADDO: Automatic Service Brokering in Service Oriented Architectures.  
URL: <http://www.vs.uni-kassel.de/research/addo/>.
- [2] Web Service Challenge.  
URL: <http://www.ws-challenge.org/>.
- [3] S. Bleul, T. Weise, and K. Geihs. Large-Scale Service Composition in Semantic Service Discovery. In *Proceedings of the IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, June 2006.
- [4] D. M. et al. *OWL-S, OWL-based Web Service Ontology*, 2004. URL: <http://www.daml.org/services/owl-s/1.1/>.
- [5] A. Gomez-Perez, R. Gonzales-Cabero, and M. Lama. ODE SWS: A Framework for Designing and Composing Semantic Web Services. In *IEEE Intelligent Systems*, pp 24-31, July, August 2004.
- [6] LSDIS Lab, University of Georgia. *METEOR-S: Semantic Web Services and Processes*, 2004.  
URL: <http://lstdis.cs.uga.edu/projects/meteor-s/>.
- [7] D. Roman, U. Keller, and H. Lausen. WSMO - Web Service Modeling Ontology. In *DERI Working Draft 14*. Digital Enterprise Research Institute (DERI), February 2004.
- [8] T. Weise and K. Geihs. DGPF - An Adaptable Framework for Distributed Multi-Objective Search Algorithms Applied to the Genetic Programming of Sensor Networks. In *Proceedings of Second International Conference on Bioinspired Optimization Methods and their Application 2006*.
- [9] T. Weise and K. Geihs. Genetic Programming Techniques for Sensor Networks. In *Proceedings of 5. GIITG KuVS Fachgespräch "Drahtlose Sensornetze"*, pages 21–25, 2006.
- [10] T. Weise, K. Geihs, and P. A. Baer. Genetic programming for proactive aggregation protocols. In B. Beliczyński, A. Dzieliński, M. Iwanowski, and B. Ribeiro, editors, *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms ICANNGA'07, Part 1*, volume 4431 of *Lecture Notes in Computer Science (LNCS)*, pages 167–173. University of Kassel, Springer Berlin Heidelberg New York, Apr 2007.
- [11] World Wide Web Consortium. *WSDL, Web Services Description Language*, August 2005.  
URL: <http://www.w3.org/TR/2005/WD-wsdl20-primer-20050803/>.

```

@inproceedings{BWG2007WSC,
author      = {Steffen Bleul and Thomas Weise and Kurt Geihs},
title       = {Making a Fast Semantic Service Composition System Faster},
booktitle   = {Proceedings of IEEE Joint Conference (CEC/EEE 2007) on E-Commerce
                Technology (9th CEC'07) and Enterprise Computing, E-Commerce and
                E-Services (4th EEE'07)},
ISBN        = {ISBN-10: 0-7695-2913-5, ISBN-13: 978-0-7695-2913-4},
pages       = {517--520},
publisher   = {IEEE Computer Society},
year        = {2007},
month       = {jul},
type        = {Conference Paper, Competition Contribution},
affiliation = {University of Kassel},
location    = {National Center of Sciences, Tokyo, Japan},
address     = {National Center of Sciences, Tokyo, Japan},
institution = {Institute of Electrical and Electronics Engineers, Inc. (IEEE)},
organization = {IEEE Computer Society},
note        = {After winning the 2006 Web Service Challenge, we now made a good second
                place in a competition with international teams from Austria, China
                (1st place), and the USA.\\
                The 2007 Web Services Challenge was held in conjunction with the IEEE
                Joint Conference (CEC/EEE 2007) on E-Commerce Technology (9th CEC'07)
                and Enterprise Computing, E-Commerce and E-Services (4th EEE'07).
                Conference Home: http://ieejc.ise.eng.osaka-u.ac.jp/CEC2007/\\
                The work is online available at
                http://www.it-weise.de/documents/index.html#BWG2007WSC.\\
                The paper can be downloaded at
                http://www.it-weise.de/documents/files/BWG2007WSC.pdf.\\
                The presentation can be downloaded at
                http://www.it-weise.de/documents/files/BWG2007WSC\_slides.pdf.\\
                The software can be downloaded at
                http://www.it-weise.de/documents/files/BWG2007WSC\_software.zip.\\
                Contact Thomas Weise at tweise@gmx.de or http://www.it-weise.de/.},
copyright   = {IEEE 2007},
abstract    = {Service Oriented Architecture (SOA) is a flexible software design
                paradigm for enterprises. The workflows of a company are implemented as
                services which can be arranged, updated and managed at runtime without
                interfering with ongoing business. Service management aims at providing
                undisturbed access to services. Its efficiency strongly depends on a
                fast response time in the case of a failure. This is hard to achieve
                since the relations between applications and services require
                comprehensive knowledge and lack transparency for administrators.
                Self-organizing approaches promise a solution by automating service
                discovery. In this paper we present a service discovery system which
                enables service compositions from semantic descriptions stored in a
                knowledge base. Therefore, it utilizes multiple composition algorithms
                from which the most appropriate set is selected and applied according
                to the size of the knowledge base and the available processors. The
                functionality of our system is made available through a Web Service
                interface itself. It is thus applicable in self-organizing service
                management systems with any number of services and ontologies.},
contents    = {* Introduction\\
                * Architectural Design\\
                * Semantic Service Composition\\
                * Implementation\\
                * Related Work\\
                * Conclusion},
keywords     = {semantics, web service, SOA, semantic web service, service composition,
                composition, discovery, ontology, taxonomy, heuristic, IDDFS, genetic
                algorithm},
language     = {en},
url          = {http://www.it-weise.de/documents/index.html#BWG2007WSC}
}

```