

Web Service Composition Systems for the Web Service Challenge – A Detailed Review

Thomas Weise, Steffen Bleul, Kurt Geihs
University of Kassel
Wilhelmshöher Allee 73
34121 Kassel, Germany
`weise|bleul|geihs@vs.uni-kassel.de`

November 20, 2007

Preview

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

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

Abstract

This report gives a detailed discussion on the system, algorithms, and techniques that we have applied in order to solve the Web Service Challenges (WSC) of the years 2006 and 2007. These international contests are focused on semantic web service composition. In each challenge of the contests, a repository of web services is given. The input and output parameters of the services in the repository are annotated with semantic concepts. A query to a semantic composition engine contains a set of available input concepts and a set of wanted output concepts. In order to employ an offered service for a requested role, the concepts of the input parameters of the offered operations must be more general than requested (contravariance). In contrast, the concepts of the output parameters of the offered service must be more specific than requested (covariance). The engine should respond to a query by providing a valid composition as fast as possible. We discuss three different methods for web service composition: an uninformed search in form of an IDDFS algorithm, a greedy informed search based on heuristic functions, and a multi-objective genetic algorithm.

1 Introduction

1.1 Web Service Composition

The necessity for fast service composition systems and the overall idea of the WS-Challenge is directly connected with the emergence of Service-Oriented Architectures (SOA). Today, companies rely on IT-architectures which are as flexible as their business strategy. The software of an enterprise must be able to adapt to changes in the business processes, regarding for instance accounting, billing, the workflows, and even in the office software. If external vendors, suppliers, or customers change, interfaces to their IT systems must be created or modified. Hence, the architecture of this software has to be built with the anticipation of changes and updates [1, 2, 3].

A SOA is the ideal architecture for such systems [4, 5]. Service oriented architectures allow us to modularize business logic and implement in the form of services accessible in a network. Services are building blocks for service processes which represent the workflows of an enterprise. Services 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.

Business now depends on the availability of service functionality, which is ensured by service management. Manual service management becomes more and more cumbersome and ineffective with a rising count of relations between services and applications. Self-organization promises a solution for finding services that offer a specific functionality automatically.

Such self-organizing approaches need a combination of syntactic and semantic service descriptions in order to decide whether a service provides a wanted functionality or not. Common syntactic definitions like WSDL [6] specify the order and types of service parameters and return values. Semantic interface description languages like OWL-S [7] or WSMO [8, 9] annotate these parameters with a meaning. Where 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 an taxonomy we can now deduce that a values which are annotated as either `ISBN-10` or `ISBN-13`¹ 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 these available input parameters and its return values contain the needed

¹There 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].

output values. In order to find such services, the semantic concepts of their parameters are matched rather than their syntactic data types.

Many service management approaches employ this semantic service discovery [10, 11, 12, 13, 8, 9, 14, 15]. 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 (*compositions*) are discovered. The sequential execution of these services provides the requested functionality.

1.2 The Web Service Challenge

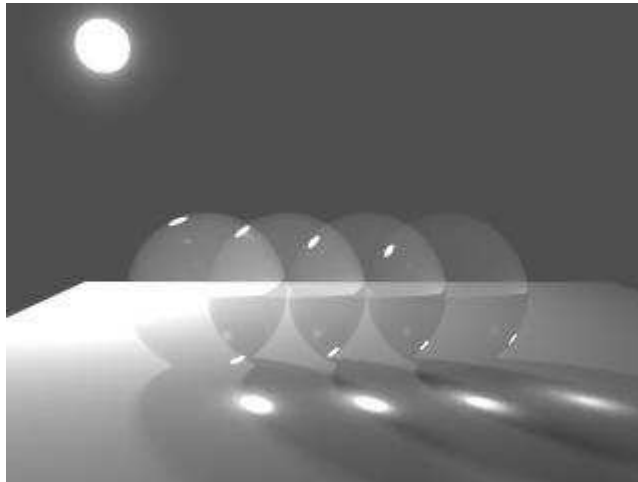


Figure 1: The logo of the Web Service Challenge.

Since 2005 the annual Web Service Challenge² (WS-Challenge, WSC) provides a platform for researchers in the area of web service composition to compare their systems and exchange experiences [16, 17, 18]. It is co-located with the IEEE Conference on Electronic Commerce (CEC) and the IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE).

Each team participating in this challenge has to provide a software system. A jury then uses these systems to solve different, complicated web service discovery and composition tasks. The major evaluation criterion for the composition systems is the speed with which the problems are solved. Another criterion is the completeness of the solution. Additionally, there is also a prize for the best overall system architecture.

²see <http://www.ws-challenge.org/> [accessed 2007-09-02]

1.3 The 2006/2007 Semantic Challenge

We have participated both in the web service 2006 and 2007 challenges [19, 20]. We managed to achieve a first place in 2006 and a second place in 2007. Here we present our system, algorithms and data structures for semantic web service composition.

The tasks of the 2006 Web Service Challenge in San Francisco, USA and the 2007 WSC in Tokyo, Japan are quite similar and only deviate in the way in which the solutions have to be provided by the software systems. Hence, we will discuss the two challenges together in this single report. Furthermore, we only consider the semantic challenges, since they are more demanding than mere syntactic matching.

2 Semantic Service Composition

In order to discuss the idea of semantic service composition properly, we need some prerequisites. Therefore, let us initially define the set of all semantic concepts \mathbb{M} . \mathbb{M} is a set of trees (taxonomies) of semantic concepts and all concepts that exist in the knowledge base are members of \mathbb{M} .

Definition 1 (subsumes) *Two concepts $A, B \in \mathbb{M}$ can be related in one of four possible ways. We define the predicate $subsumes(X, Y) : X, Y \in \mathbb{M}$ 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 is a generalization of B ($subsumes(A, B)$) and B is a generalization of C ($subsumes(B, C)$), then A is also a generalization of C ($subsumes(A, C)$). The same goes vice versa for specialization, here we can define that if A is a specialization of B ($subsumes(B, A)$) and A is also a specialization of C ($subsumes(C, A)$), then either $subsumes(B, C)$ or $subsumes(C, B)$ (or both) must hold, i. e. either C is a specialization of B , or B is a specialization of C , or $B = C$.

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 . The hierarchy defined here is pretty much

the same as in object-oriented programming languages. If we imagine \mathbf{A} and \mathbf{B} to be classes in Java, $subsumes(A, B)$ can be considered to be equivalent to the expression `A.class.isAssignableFrom(B.class)`. If it evaluates to `true`, a value y of type \mathbf{B} can be assigned to a variable x of type \mathbf{A} since y `instanceof` \mathbf{A} will hold.

From the viewpoint of a composition algorithm, there is no need for a distinction between parameters and the annotated concepts. The set of services \mathbb{S} contains all the services s known to the service registry. Each service $s \in \mathbb{S}$ has a set of required input concepts $s.in \subseteq \mathbb{M}$ which it expects when being executed 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. After its completion, the service will return a set of output parameters as defined in its interface description.

Similar, 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 (partially³) ordered set of n services $S = \{s_1, s_2, \dots, s_n\} : s_1, \dots, s_n \in \mathbb{S}$ that can successively be executed with the accumulated input parameters so that output parameters produced by these services are treated as available input parameters in the next execution step. S can be executed with the available input parameters defined in $R.in$. If it is executed, it produces outputs that are either annotated with exactly the requested concepts $R.out$ or with more specific ones (*covariance*). This is the case if they can be brought into an order (s_1, s_2, \dots, s_n) in a way that

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

assuming that $R.in \cap R.out = \emptyset$ in the last line of the equation. With equation 1 we have defined the goal predicate which we can use in any form of informed or uninformed state space search.

3 The Problem Definition

In the 2006 and 2007 WSC, the composition software is provided with three parameters:

1. A concept taxonomy to be loaded into the knowledge base of the system. This taxonomy was stored in a file of the XML Schema format [21].
2. A directory containing the specifications of the service to be loaded into the service registry. For each service, there was a single file given in WSDL format [6].

³The set S is only partially ordered since, in principle, some services may be executed in parallel if they do not depend on each other.

3. A query file containing multiple service composition requests R_1, R_2, \dots in a made-up XML [22] format.

These formats are very common and allow the contestants to apply the solutions in real world applications later, or to customize their already existing applications so they can be used as contribution in the competition.

3.1 Semantic Concepts and Taxonomies

In Section 2 we have discussed that the subsumes relation between two semantic concepts can be compared to the subclass relationship in object-oriented programming. In the WSC, exactly this analogue is used: Concepts are treated as data types and taxonomies can be encoded as hierarchies of such data types in XSD schemas.

```
1 <complexType name="Address">
2   <sequence>
3     <element name="name" type="string" minOccurs="0"/>
4     <element name="street" type="string"/>
5     <element name="city" type="string"/>
6   </sequence>
7 </complexType>
8
9 <complexType name="US-Address">
10  <complexContent>
11    <extension base="Address">
12      <sequence>
13        <element name="state" type="US-State"/>
14        <element name="zip" type="positiveInteger"/>
15      </sequence>
16    </extension>
17  </complexContent>
18 </complexType>
```

Listing 1: Example for an XSD schema definition.

Listing 1 shows a sample XSD schema defining the data types `Address` and `US-Address` inheriting from `Address`. In the context of the WSC, this schema would be interpreted as a taxonomy introducing the concepts `Address` and `US-Address` with $subsumes(Address, US-Address)$.

3.2 Interface Specifications

The interfaces of web services are specified with WSDL in the Web Service Challenge. The input and output messages of the services may contain multiple parameters. Each parameter is annotated with a semantic concept stored in the attribute `type`.

```
1 <message name="InputName">
2   <part name="part0" type="Name"/>
3 </message>
4 <message name="OutputAddress">
5   <part name="part0" type="US-Address"/>
```

```

6 </message>
7 <portType name="AdressConverter">
8   <operation name="Convert">
9     <input message="InputName"/>
10    <output message="OutputAddress"/>
11  </operation>
12 </portType>

```

Listing 2: Example for a WSDL service interface description.

In listing 2, the service `AddressConverter` defines one operation named `Convert`. It can be invoked with an input message (`input`-tag) and produces a response message (`output`). The value of the attribute `message` represents a reference to a `message` element. Each `message` has a set of `part` elements as children which represent the service parameters, annotated with concepts referenced by the `type`-attribute. The `Convert` operation in this example requires a parameter of the type `Name` and returns an instance of `US-Address`.

3.3 Result Format

The expected result to be returned by the software was also a stream of data in a proprietary XML dialect containing all possible service compositions that solved the queries according to equation 1. We will not discuss the data formats used in this challenge any further since they are replaceable and do not contribute to the way the composition queries are solved. It was possible that a request R_i was resolved by multiple service compositions. In the 2006 challenge, the communication between the jury and the programs was via command line or other interfaces provided by the software, in 2007 a web service interface was obligatory.

Remarkably are furthermore some of the restrictions in the challenge tasks:

- There exists at least one solution for each query.
- The services in the solutions are represented as a sequence of sets. Each set contains equivalent services. Executing one service from each set forms a valid composition S . This representation does not allow for any notation of parallelization.

These restrictions sure will be removed in future WSCs.

Before we elaborate on the solution itself, let us define the operation *getPromisingServices* which obtains the set $V \subseteq \mathbb{S}$ of services $s \in \mathbb{S}$ that produce an output parameter annotated with the concept A (regardless of their inputs).

$$\forall s \in \text{getPromisingServices}(A) \exists B \in s.out : \text{subsumes}(A, B) \quad (2)$$

The composition system that we have applied in the 2007 WSC consists of three types of composition algorithms. The search space \tilde{X} that they investigate is basically the set of all possible permutations of all possible sets

of services. The power set $\mathcal{P}(\mathbb{S})$ includes all possible subsets of \mathbb{S} . \tilde{X} is then the set of all possible permutations of the elements in such subsets, in other words $\tilde{X} \subseteq \{\forall perm(\xi) : \xi \in \mathcal{P}(\mathbb{S})\}$.

4 An (Uninformed) Algorithm Based on ID-DFS

Our first solution approach is based on an uninformed search. Therefore, we first discuss uninformed searches before specifying the composition algorithm.

4.1 Uninformed Search

The simplest form of search algorithm is the uninformed search⁴. It does not rely on any information about the structure of a possible solution. Generally, only two operations must be defined in such search algorithms: one that tells us if we have found what we are looking for (*isGoal*) and one that helps enumerating the search space (*expand*) [23].

Definition 2 (isGoal) *The function $isGoal(x) \in \{true, false\}$ is the target predicate of state space search algorithms that tells us whether a given state $x \in \tilde{X}$ is a valid solution (by returning *true*), i. e. the goal state, or not (by returning *false*).*

Definition 3 (expand) *The operator $expand(x)$ computes a set of solution candidates (states) X_{new} from a given state x . It is the exploration operation of state space search algorithms. Different from the mutation operator of evolutionary algorithms (see Section 6.1.2 on page 18), it is strictly deterministic and returns a set instead of single individual. Applying it to the same x values will always yield the same set X_{new} .*

Uninformed search algorithms are very general and can be applied to a wide variety of problems. Their common drawback is that search spaces are often very large. Without the incorporation of information, for example in form of heuristic functions, the search may take very long and quickly becomes infeasible [24, 25, 26].

4.1.1 Depth-limited Search

The depth-limited search⁵ [24] is a depth-first search (DFS) that only proceeds up to a given maximum depth d . In other words, it does not examine solution candidates that are more than d *expand*-operations away from the root state r , as outlined in algorithm 1. Analogously to the plain depth first search, the time complexity is b^d and the memory complexity is in $\mathcal{O}(b * d)$. Of course, the depth-limited search can neither be complete nor optimal. If a maximum depth of the possible solutions however known, it may be sufficient.

⁴http://en.wikipedia.org/wiki/Uninformed_search [accessed 2007-08-07]

⁵http://en.wikipedia.org/wiki/Depth-limited_search [accessed 2007-08-07]

Algorithm 1: $S = dl_dfs(r, d)$

Input: $r \in \tilde{X}$ the node to be explored

Input: $d \in \mathbb{N}$ the maximum depth

Input: Implicit: $expand$ the expansion operator

Input: Implicit: $isGoal$ an operator that checks whether a state is a goal state or not

Data: $x \in \tilde{X}$ the state currently processed

Data: $X \in \tilde{X}$ set of “expanded” states

Output: $S \in \tilde{X}$ the solution state found, or \emptyset

```
1 begin
2   if  $isGoal(r)$  then return  $\{r\}$ 
3   if  $d > 0$  then
4     foreach  $x \in expand(r)$  do
5        $X \leftarrow dl\_dfs(x, d - 1)$ 
6       if  $X \neq \emptyset$  then return  $X$ 
7   return  $\emptyset$ 
8 end
```

4.1.2 Iterative deepening depth-first search

The iterative deepening depth-first search⁶ (IDDFS, [24]), defined in algorithm 2, iteratively runs a depth-limited DFS with stepwise increasing maximum depths d . In each iteration, it visits the states in the state space according to the depth-first search. Since the maximum depth is always incremented by one, one new level (in terms means of distance in $expand$ operations from the root) is explored in each iteration. This effectively leads to a breadth-first search.

IDDFS thus unites the advantages of breadth-first search and death-first search: It is complete and optimal, but only has a linearly rising memory consumption in $\mathcal{O}(d * b)$. The time consumption, of course, is still in $\mathcal{O}(b^d)$. IDDFS is the best uninformed search strategy and can be applied to large search spaces with unknown depth of the solution.

4.2 The IDDFS Composition Algorithm

Our first composition algorithm uses such an iterative deepening depth-first search. It is only fast in finding solutions for small service repositories but optimal if the problem requires an exhaustive search. Thus, it may be used by the strategic planner in conjunction with another algorithm that runs in parallel if the size of the repository is reasonable small.

Algorithm 3 (*webServiceCompositionIDDFS*) builds a valid web service composition starting from the back. In each recursion, its internal helper

⁶<http://en.wikipedia.org/wiki/IDDFS> [accessed 2007-08-08]

Algorithm 2: $S = iddfs(r)$

Input: $r \in \tilde{X}$ the node to be explored
Input: Implicit: *expand* the expansion operator
Input: Implicit: *isGoal* an operator that checks whether a state is a goal state or not
Data: $d \in \mathbb{N}$ the (current) maximum depth
Output: $S \in \tilde{X}$ the solution state found, or \emptyset

```
1 begin
2    $d \leftarrow 0$ 
   /* This algorithm is for infinitely large search spaces. In
   real systems, there is a maximum  $d$  after which the whole
   space would be explored and the algorithm should return  $\emptyset$ 
   if no solution was found. */
3   repeat
4      $S \leftarrow dl\_dfs(r, d)$ 
5      $d \leftarrow d + 1$ 
6   until  $S \neq \emptyset$ 
7   return  $S$ 
8 end
```

method *dl_dfs_wsc* tests all elements A of the set *wanted* of yet unknown parameters. It then iterates over the set all services s that can provide A . For every single s *wanted* is recomputed. If it becomes the empty set \emptyset , we have found a valid composition and can return it. If *dl_dfs_wsc* is not able to find a solution within the maximum depth limit (which denotes the maximum number of services in the composition), it returns \emptyset . The loop in *webServiceCompositionIDDFS* iteratively invokes *dl_dfs_wsc* by increasing the depth limit step by step, until a valid solution is found.

5 An (Informed) Heuristic Approach

The IDDFS-algorithm just discussed performs an uninformed search in the space of possible service compositions. It is slow and memory consuming for bigger repositories since it does not utilize any additional information about the search space. If we use such information, we can increase the efficiency of the search remarkably.

5.1 Informed Search

In an informed search⁷, a heuristic function helps to decide which nodes are to be expanded next. If the heuristic is good, informed search algorithms may

⁷http://en.wikipedia.org/wiki/Search_algorithms#Informed_search [accessed 2007-08-08]

Algorithm 3: $S = \text{webServiceCompositionIDDFS}(R)$

Input: R the composition request

Data: maxDepth , depth the maximum and the current search depth

Data: in , out current parameter sets

Data: composition , comp the current compositions

Data: A, B, C, D, E some concepts

Output: S a valid service composition solving R

```
1 begin
2    $\text{maxDepth} \leftarrow 2$ 
3   repeat
4      $S \leftarrow \text{dl\_dfs\_wsc}(R.\text{in}, R.\text{out}, \emptyset, 1)$ 
5      $\text{maxDepth} \leftarrow \text{maxDepth} + 1$ 
6   until  $S \neq \emptyset$ 
7 end

8  $\text{dl\_dfs\_wsc}(\text{in}, \text{out}, \text{composition}, \text{depth})$ 
9 begin
10  foreach  $A \in \text{out}$  do
11    foreach  $s \in \text{getPromisingServices}(A)$  do
12       $\text{wanted} \leftarrow \text{out}$ 
13      foreach  $B \in \text{wanted}$  do
14        if  $\exists C \in s.\text{out} : \text{subsumes}(B, C)$  then
15           $\text{wanted} \leftarrow \text{wanted} \setminus \{B\}$ 
16        foreach  $D \in s.\text{in}$  do
17          if  $\nexists E \in \text{in} : \text{subsumes}(D, E)$  then
18             $\text{wanted} \leftarrow \text{wanted} \cup \{D\}$ 
19           $\text{comp} \leftarrow s \oplus \text{composition}$ 
20          if  $\text{wanted} = \emptyset$  then
21            return  $\text{comp}$ 
22          else
23            if  $\text{depth} < \text{maxDepth}$  then
24               $\text{comp} \leftarrow \text{dl\_dfs\_wsc}(\text{in}, \text{wanted}, \text{comp}, \text{depth} + 1)$ 
25            if  $\text{comp} \neq \emptyset$  then return  $\text{comp}$ 
26          end
27        end
28      end
29    end
30  end
31  return  $\emptyset$ 
32 end
```

dramatically outperform uninformed strategies [27, 28, 29, 23].

Heuristic functions are problem domain dependent. In the context of an informed search, a heuristic function $h : \tilde{X} \mapsto \mathbb{R}^+$ maps the states in the state space \tilde{X} to the positive real numbers \mathbb{R}^+ . The value $h(s)$ should be some form of *estimate* on how likely expanding or testing the state s will lead to a correct solution or how many expansion steps a correct solution is away. Here we focus on the latter notation which makes heuristics subject to minimization. This also means that all heuristics become zero if s already is a valid solution.

$$\forall s \in \tilde{X} : isGoal(s) \Rightarrow h(s) = 0 \quad \forall \text{heuristics } h : \tilde{X} \mapsto \mathbb{R}^+ \quad (3)$$

Since the value of a heuristic function $h(s)$ for a state s is the higher, the more *expand*-steps s is *probably* (or *approximately*) away from a valid solution, it represents the distance of an individual to a solution in *solution space*.

A best-first search⁸ is a search algorithm that incorporates such an estimation function v in a way that promising solution candidates s with low estimation values $v(s)$ are evaluated before other states t that receive a higher values $v(t) > v(s)$. For estimation functions, the same constraints are valid as for heuristic functions. Matter of fact, an estimation may be a heuristic function itself (as in greedy search) or be based on a heuristic function (as in A* search).

5.1.1 Greedy Search

A greedy search⁹ is a best-first search where the currently known solution candidate with the lowest heuristic value is investigated next. Here, the estimation function is the heuristic function itself.

The greedy algorithm internal sorts the list of currently known states in *descending* order according to a comparator function $c_h(x_1, x_2) \in \mathbb{R}$. As a comparator function, c_h will be below zero if x_1 should be preferred instead of x_2 ($h(x_1) < h(x_2)$) and higher then zero for all $h(x_1) > h(x_2)$, which indicate that x_2 is more a more prospective solution candidate. Thus, the elements with the best heuristic value will be at the end of the list, which then can be used as a stack.

$$c_h(x_1, x_2) = h(x_1) - h(x_2) \quad (4)$$

The greedy search as specified in algorithm 4 now works like a depth-first search on this stack. It thus also shares most of the properties of the DFS. It is neither complete nor optimal and its worst case time consumption is b^m . On the other hand, like breadth-first search, its worst-case memory consumption is also b^m .

Notice that we can replace c_h with any other valid comparator function. In principle, we could even apply objective functions and Pareto-based comparisons (which are discussed in detail in Section 6.1.1 on page 17).

⁸http://en.wikipedia.org/wiki/Best-first_search [accessed 2007-09-25]

⁹http://en.wikipedia.org/wiki/Greedy_search [accessed 2007-08-08]

Algorithm 4: $S = greedySearch(r)$

Input: $r \in \tilde{X}$ the root node to start the expansion at
Input: Implicit: $h : \tilde{X} \mapsto \mathbb{R}^+$ the heuristic function
Input: Implicit: $expand$ the expansion operator
Input: Implicit: $isGoal$ an operator that checks whether a state is a goal state or not
Data: $x \in \tilde{X}$ the state currently processed
Data: $X \in \tilde{X}$ the sorted list of states to explore
Output: $S \in \tilde{X}$ the solution state found, or \emptyset

```
1 begin
2    $X \leftarrow (r)$ 
3   while  $X \neq \emptyset$  do
4      $X \leftarrow sort_d(X, c_h)$ 
5      $x \leftarrow deleteListItem(X, |X| - 1)$ 
6     if  $isGoal(x)$  then return  $x$ 
7      $X \leftarrow appendList(X, expand(x))$ 
8   return  $\emptyset$ 
9 end
```

5.2 The Greedy Composition Algorithm

We can easily derive heuristic functions for the area of web service composition. Therefore, we will again need some further definitions. Notice that the set functions specified in the following does not need to be evaluated every time they are queried, since we can maintain their information as meta-data along with the composition and thus save runtime.

Let us first define the set of unsatisfied parameters $wanted(S) \subseteq \mathbb{M}$ in a candidate composition S as

$$A \in wanted(S) \Leftrightarrow (\exists s \in S : A \in s.in \vee A \in R.out) \wedge A \notin R.in \bigcup_{i=1}^{|S|} s_i \quad \dots (s_i \in S) \quad (5)$$

In other words, a wanted parameter is either an output concept of the composition query or an input concept of any of the services in the composition candidate that hasn't been satisfied by neither an input parameter of the query nor by an output parameter of any service. Here we assume that the concept A wanted by service s is not also an output parameter of s . This is done for simplification purposes – the implementation has to keep track of this possibility.

The set of *eliminated* parameters of a service composition contains all input parameters of the services of the composition and queried output pa-

rameters of the composition request that already have been satisfied.

$$eliminated(S) = \left[R.out \cup \left(\bigcup_{i=1}^{|S|} s_i.in \right) \right] \setminus wanted(S) \quad (6)$$

Finally, the set of *known* concepts is the union of the input parameters defined in the composition request and the output parameters of all services in the composition candidate.

$$known(S) = R.in \cup \bigcup_{i=1}^{|S|} s_i.out \quad (7)$$

Instead of using these sets to build a heuristic, we can derive a comparator function c_{wsc} directly (see Section 5.1.1 on page 12). This comparator function has the advantage that we also can apply randomized optimization methods like evolutionary algorithms based on it.

Algorithm 5: $r = c_{wsc}(S_1, S_2)$

Input: $S_1, S_2 \in \tilde{X}$ two composition candidates
Data: i_1, i_2, e_1, e_2 some variables
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  $i_1 \leq 0$  then
5     if  $i_2 \leq 0$  then return  $|S_1| - |S_2|$ 
6     else return -1
7   if  $i_2 \leq 0$  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
12    if  $e_1 < e_2$  then return 1
13  if  $i_1 > i_2$  then return -1
14  else
15    if  $i_1 < i_2$  then return 1
16  if  $|S_1| \neq |S_2|$  then return  $|S_1| - |S_2|$ 
17  return  $|known(S_1)| - |known(S_2)|$ 
18 end
```

Algorithm 5 defines c_{wsc} which compares two composition candidates S_1 and S_2 . This function can be used by a greedy search algorithm in order to

decide which of the two possible solutions is more prospective. c_{wsc} will return a negative value if S_1 seems to be closer to a solution than S_2 , a positive value if S_2 looks as if it should be examined before S_1 , and zero if both seem to be equally good.

The first thing it does is comparing 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. If the comparator has not returned a value yet, it means that both candidates still have wanted concepts. For us, it was surprising that it is better to use the number of already satisfied concepts as next comparison criterion instead of the number of remaining unsatisfied concepts. However, if we do so, the search algorithms perform significantly faster. Only if both composition candidates 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 the compositions are even of the same length, we finally base the decision on the total number of known concepts.

The form of this interesting comparator function is maybe caused by the special requirements of the WSC data. Nevertheless, it shows which sorts of information about a composition can be incorporated into the search.

The interesting thing that we experienced in our experiments is that it is not a good idea to decide on the utility of a solution candidate with

In order to apply pure greed search, we still need to specify the *expand* operator computing the set of possible offspring that can be derived from a given individual. In algorithm 3, we have realized it implicitly. Additionally, we can also define the *isGoal* predicate on basis of the *wanted* function:

$$\text{expand}(S) \equiv s \oplus S \forall s, A : s \in \text{getPromisingServices}(A) \wedge A \in \text{wanted}(S) \quad (8)$$

$$\text{isGoal}(S) \equiv \text{wanted}(S) = \emptyset \quad (9)$$

With these definitions, we can now employ plain greedy search as defined in algorithm 4 on page 13.

6 A Genetic Approach

6.1 Evolutionary Algorithms

Evolutionary algorithms (EA) [30, 31, 32, 23] are generic, population-based meta-heuristic optimization algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection and survival of the fittest.

The family of evolutionary algorithms encompasses genetic algorithms (see Section 6.1.2), genetic programming, evolution strategy, evolutionary programming, and learning classifier systems.

The advantage of evolutionary algorithms compared to other optimization methods is that they make only few assumptions about the underlying fitness

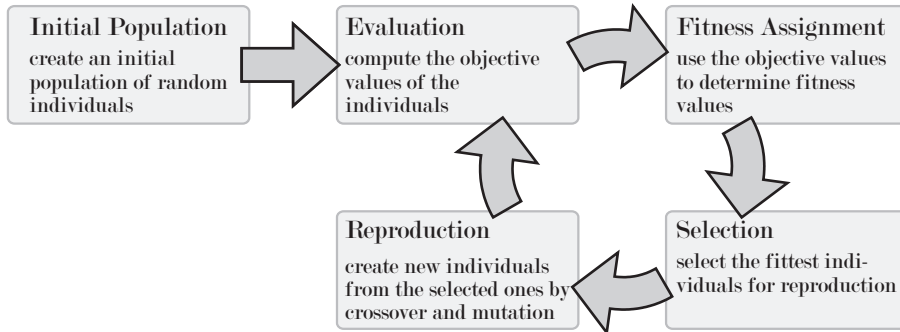


Figure 2: The basic cycle of evolutionary algorithms.

landscape and therefore perform consistently well in many different problem categories.

All evolutionary algorithms proceed in principle according to the scheme illustrated in Figure 2:

1. Initially, a population of individuals with a totally random genome is created.
2. All individuals of the population are tested. This evaluation may incorporate complicated simulation and calculations.
3. With the tests, we have determined the utility of the different features of the solution candidates and can now assign a fitness value to each of them.
4. A subsequent selection process filters out the individuals with low fitness and allows those with good fitness to enter the mating pool with a higher probability.
5. In the reproduction phase, offspring is created by varying or combining these solution candidates and integrated it into the population.
6. If the termination criterion is met, the evolution stops here. Otherwise, it continues at step 2.

In the context of this report, especially genetic algorithms are of interest.

6.1.1 Multi-Objective Optimization

We can furthermore distinguish between single-objective and multi-objective evolutionary algorithms (MOEA), where the latter means that we try to optimize multiple, possible conflicting criteria. Our following elaborations will be based on MOEAs. The general area of evolutionary computation that

deals with multi-objective optimization is called EMOO, evolutionary multi-objective optimization.

Definition 4 (MOEA) *A multi-objective evolutionary algorithm (MOEA) is able to perform an optimization of multiple criteria on the basis of artificial evolution [33, 34, 35, 36, 37, 38, 39].*

Multi-objective optimization is applied whenever there are different, conflicting goals to be achieved with the optimization process. Our web service composition for example should on one hand be short and on the other hand complete.

Pareto Optimization Pareto efficiency¹⁰, or Pareto optimality, is an important notion in neoclassical economics with broad applications in game theory, engineering and the social sciences [40, 41].

It defines the front of solutions that can be reached by trading-off conflicting objectives in an optimal manner. From this front, a decision maker (be it a human or another algorithm) can finally choose the configuration that, in his opinion, suits best [42, 43, 44, 45, 46].

The notation of Pareto optimal is strongly based on the definition of domination:

Definition 5 (Domination) *An element x_1 dominates (is preferred to) an element x_2 ($x_1 \vdash x_2$) if x_1 is better than x_2 in at least one objective function and not worse with respect to all other objective functions $f \in F$.*

$$x_1 \vdash x_2 \Leftrightarrow \begin{aligned} &\forall i : 0 < i \leq n \Rightarrow \omega_i f_i(x_1) \geq \omega_i f_i(x_2) \wedge \\ &\exists j : 0 < j \leq n : \omega_j f_j(x_1) > \omega_j f_j(x_2) \end{aligned} \quad (10)$$

$$\omega_i = \begin{cases} -1 & \text{if } f_i \text{ should be minimized} \\ 1 & \text{if } f_i \text{ should be maximized} \end{cases} \quad (11)$$

The Pareto domination relation defines a partial order on the set of possible objective values.

6.1.2 Genetic Algorithms

Genetic algorithms are a subclass of evolutionary algorithms that employs two different representations for each solution candidate. The genotype is used in the reproduction operations whereas the phenotype is the form of the individual which can be used for the determining its fitness [47, 48, 49, 50, 51, 23].

¹⁰http://en.wikipedia.org/wiki/Pareto_efficiency [accessed 2007-07-03]

Mutation Mutation is an important method of preserving individual diversity. In fixed-length string chromosomes it can be achieved by modifying the value of one element of the genotype, as illustrated in Figure 3. More generally, a mutation may change $0 \leq n < |g|$ locations in the string. In binary coded chromosomes for example, the elements are bits which are simply toggled. If the genetic algorithm consists of genes that for example identify a service, such a modification would lead to replacing this service with another one.

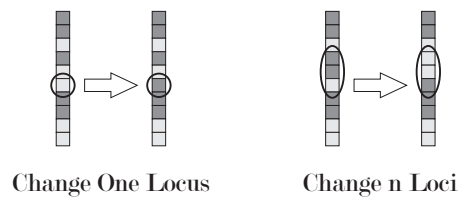


Figure 3: Bit-toggling mutation of string chromosomes.

If the string chromosomes are of variable length, the set of mutation operations can be extended by two additional methods. On one hand, one could insert a couple of elements at any given position into a chromosome. On the other hand, this operation can be reversed by deleting elements from the string. It should be noted that both, insertion and deletion, are also implicitly performed by crossover. Recombining two identical strings with each other can for example lead to deletion of genes. The crossover of different strings may turn out as an insertion of new genes into an individual.

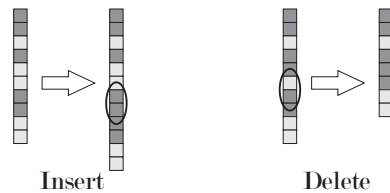


Figure 4: Mutation of variable-length string chromosomes.

Crossover Figure 5 outlines the recombination of two string chromosomes. This operation is called crossover and is done by swapping parts of the genotypes between the parents.

When performing crossover, both parental chromosomes are split at randomly determined crossover points. Subsequently, a new child chromosome is created by appending the first part of the first parent with the second part of the second parent. This method is called one-point crossover. In two-point

crossover, both parental genotypes are split at two points, constructing a new offspring by using parts number one and three from the first, and the middle part from the second ancestor. The generalized form of this technique is n -point crossover.

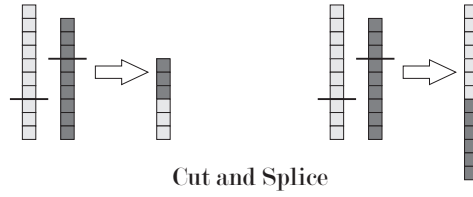


Figure 5: Crossover (recombination) of variable-length string chromosomes.

6.2 A Genetic Composition Algorithm

In order to use a genetic algorithm to breed web service compositions, we first need to define a proper genome able to represent service sequences. A straightforward yet efficient way is to use (variable-length) strings of service identifiers which can be processed by standard genetic algorithms (see Section 6.1.2 on page 17). Because of this well-known string form, we also could apply standard creation, mutation, and crossover operators.

However, by specifying a specialized mutation operation we can make the search more efficient. This new operation either deletes the first service in S (via $mutate_1$) or adds a promising service to S (as done in $mutate_2$). Using the adjustable variable σ as a threshold we can tell the search whether it should prefer growing or shrinking the solution candidates.

$$mutate_1(S) \equiv \begin{cases} \{s_2, s_2, \dots, s_{|S|}\} & \text{if } |S| > 1 \\ S & \text{otherwise} \end{cases} \quad (12)$$

$$mutate_2(S) \equiv s \oplus S : \begin{matrix} s \in getPromisingServices(A) \wedge \\ A \in wanted(S) \end{matrix} \quad (13)$$

$$mutate(S) \equiv \begin{cases} mutate_1(S) & \text{if } random_u() > \sigma \\ mutate_2(S) & \text{otherwise} \end{cases} \quad (14)$$

A new *create* operation for building the initial random configurations can be defined as a sequence of $mutate_2$ invocations of random length. Initially, $mutate_2(\emptyset)$ is invoked and will return a composition consisting of a single service that satisfies at least one parameter in $R.out$. We iteratively apply $mutate_2$ to its previous result a random number of times, in order to a new individual.

6.2.1 The Comparator Function and Pareto Optimization

As driving force for the evolutionary process we can reuse the comparator function c_{wsc} as specified as for the greedy search in algorithm 5 on page 14. It combines multiple objectives, putting pressure towards the direction of

- compositions which are complete,
- small compositions,
- compositions that resolve many unknown parameters, and
- compositions that provide many parameters.

On the other hand, we could as well separate these single aspects into different objective functions and apply direct Pareto optimization. This has the drawback that it spreads the pressure of the optimization process over the complete Pareto frontier.

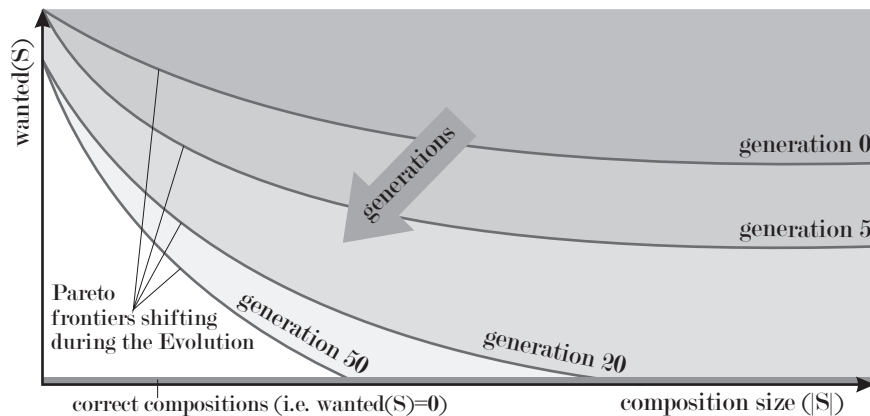


Figure 6: A sketch of the Pareto front in the genetic composition algorithm.

Figure 6 visualizes the multi-objective optimization problem “web service composition” by sketching a characteristic example for Pareto frontiers of several generations of an evolutionary algorithm. We concentrate on the two dimension *composition size* and *number of wanted (unsatisfied) parameters*. Obviously, we need to find compositions which are correct, i. e. where the latter objective is zero. On the other hand, an evolution guided only by this objective can (and will) produce compositions containing additional, useless invocations of services not related to the problem at all. The first objective is thus also required. In Figure 6, the first five or so generations are not able to produce good compositions yet. We just can observe that longer compositions tend to provide more parameters (and have thus a lower number of wanted parameters). In generation 20, the Pareto frontier is pushed farther

forward and touches the abscissa – the first correct solution is found. In the generations to come, this solution is improved and useless service calls are successively removed, so the composition size decreases. There will be a limit, illustrated as generation 50, where the shortest compositions for all possible values of *wanted* are found. From now on, the Pareto front cannot progress any further and the optimization process has come to a rest.

As you can see, pure Pareto optimization does not only seek for the best correct solution but also looks for the best possible composition consisting of only one service, for the best one with two service, with three services, and so on. This spreading of the population of course slows down the progress into the specific direction where $wanted(S)$ decreases.

The comparator function c_{wsc} has proven to be more efficient in focusing the evolution on this part of the search space. It plays the role of the external decision maker in the multi-objective optimization scheme defined by Fonseca and Fleming [52, 38, 53, 54, 55, 23]. The genetic algorithm based on this function is superior in performance and hence, is used in our experiments.

7 Experimental Results

In Table 1 we illustrate the times that the different algorithms introduced in this report needed to perform composition tasks of different complexity¹¹. We have repeated the experiments multiple times on an off-the-shelf PC¹² and noted the mean values. The times themselves are not so important, rather are the proportions and relations between them.

The IDDFS approach can only solve smaller problems and becomes infeasible very fast. When building simpler compositions though, it is about as fast as the heuristic approach, which was clearly dominating in all categories. Because a heuristic may be misleading and (although it didn't happen in our experiments) could lead to a very long computation time in the worst case. Thus we decided to keep the IDDFS and heuristic approach in our system and run them in parallel on each task if sufficient CPUs are available.

The genetic algorithm (population size 1024) was able to resolve all composition requests correctly for all knowledge base and all registry sizes. It was able to build good solutions regardless how many services had to be involved in a valid solution (solution depth). In spite of this correctness, it always was a magnitude slower than the greedy search which provided the same level of correctness.

Despite of this, the idea of applying a genetic algorithm to web service composition is maybe not so bad. If the compositions would become more complicated or would involve quality of service (QoS) aspects, it is not clear

¹¹The test sets used here are available at http://www.it-weise.de/documents/files/BWG2007WSC_software.zip [accessed November 20, 2007]. Well, at least partly, I've accidentally deleted set 12 and 13. Sorry.

¹²2 GHz, Pentium IV single core with Hyper-Threading, 1 GiB RAM, Windows XP, Java 1.6.0..03-b05

Table 1: Experimental results for the web service composers.

Test	Depth of Solution	No. of Concepts	No. of Services	IDDFS (ms)	Greedy (ms)	GA (ms)
1	5	56210	1000	241	34	376
2	12	56210	1000	-	51	1011
3	10	58254	10000	-	46	1069
4	15	58254	2000	-	36	974
5	30	58254	4000	-	70	6870
6	40	58254	8000	-	63	24117
7	1	1590	118	≤ 16	≤ 16	290
8.1	2	15540	4480	≤ 16	≤ 16	164
8.2	2	15540	4480	≤ 16	≤ 16	164
8.3	2	15540	4480	≤ 16	≤ 16	164
8.4	2	15540	4480	≤ 16	≤ 16	234
8.5	3	15540	4480	≤ 16	≤ 16	224
8.6	3	15540	4480	≤ 16	≤ 16	297
8.7	4	15540	4480	18	24	283
8.8	3	15540	4480	≤ 16	≤ 16	229
8.9	2	15540	4480	≤ 16	≤ 16	167
11.1	8	10890	4000	-	31	625
11.3	2	10890	4000	-	21	167
11.5	4	10890	4000	22021	≤ 16	281
12.1	5	43680	2000	200320	≤ 16	500
12.3	7	43680	2000	99	31	375
13	6	43680	2000	250	32	422

if these can be resolved with a simple heuristic with the same efficiency that we could observe in our greedy search approach.

8 Architectural Considerations

In 2007, we introduced a more refined version [20] of our 2006 semantic composition system [19]. The architecture of this composer, as illustrated in Figure 7, is designed in a very general way, making it not only a challenge contribution but also part of the ADDO web service brokering system [12, 10, 11]: 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. The web service composer is available for any system where semantic service discovery with the Ontology Web Language for Services (OWL-S) [7] or similar languages is used. Hence, this contest application is indeed also a real-world application.

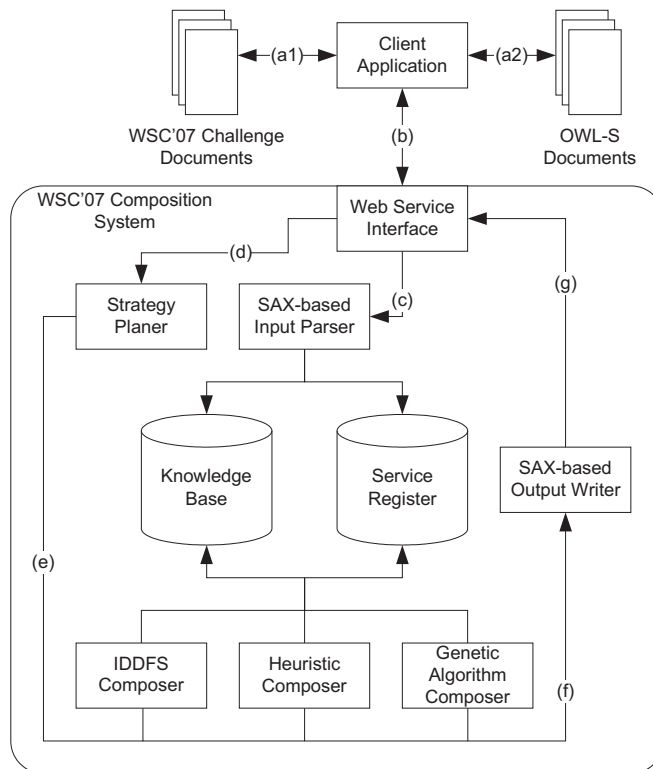


Figure 7: The WSC 2007 Composition System of Bleul and Weise.

An application accesses the composition system by submitting a service

request (illustrated by (b)) through its *Web Service Interface*. It furthermore provides the services 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. One or more composition algorithm modules solve the composition requests and pass the solution to a *SAX-based Output Writer*, an XML document generating module (f) faster than DOM serialization. Here it is also possible to transform it to, for example, BPEL4WS [56] descriptions. The result is afterwards returned through the web service Interface (g).

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. Therefore, we transparently internally merge the knowledge base and the service registry. This step is described here because it is very crucial for the overall system performance.

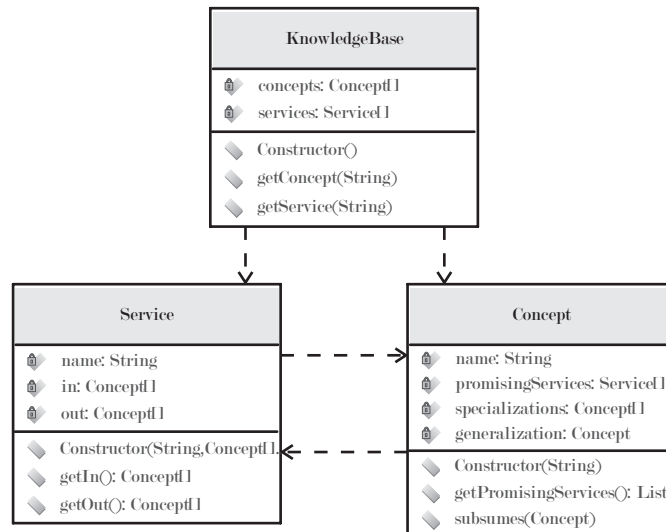


Figure 8: The Knowledge Base and Service Registry of our Composition System.

A semantic concept is represented by an instance of the class `Concept`. 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 8, 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. Each instance *x* of `Concept` holds an internal variable `promisingServices` which is initially `null`. If `x.getPromisingServices()` is invoked, it first looks up if `x.promisingServices` is `null`. If so, the list of promising services is computed, stored in `x.promisingServices`, and returned. Otherwise, `x.promisingServices` is returned directly. Since we do not synchronize this method, it may be possible that the list is computed concurrently multiple times. Each of these computations will produce the same result. Although all parallel invocations of `x.getPromisingServices()` will return other lists, their content is the same. The result of the computation finishing last will remain `x.promisingServices` whereas the other lists will get lost and eventually be freed by the garbage collector. Further calls to `x.getPromisingServices()` always will yield the same, lastly stored, result. This way, we can perform caching which is very important for the performance and spare costly synchronization while still granting a maximum degree of parallelization.

9 Related Work

9.1 Indexing and Efficient Parsing

As we mentioned in our architectural considerations section, efficient parsing, indexing, and data representation are vital for being competitive in the WSC. Other contestants made similar efforts in order to tune the performance of their systems.

9.1.1 VitaLab

The *VitaLab* [57] approach for instances uses the Streaming API for XML, StAX¹³ which is something like a pull-parsing variant of SAX. For each concept, indexing is done by creating lists of services that consume or produce in VitaLab. In our approach, only the lists providing services are needed and they are created on demand rather than building it on bootstrap. In the 2007 extension of this work a greedy search approach was used [58]. Here, the degree to which the input parameters of the query match the already available ones found by the composition algorithm serves as heuristic function.

¹³<http://www.xml.com/pub/a/2003/09/17/stax.html> [accessed 2007-10-25]

9.1.2 SWSDS

Another index-based approach is the ¹⁴ composition system [59]. It can be used for both, syntactic and semantic matching by simply switching the search index. SWSDS uses a composition algorithm similar to our uninformed ID-DFS variant (see algorithm 3 on page 11) but extended with the constraint that each service is only considered one time to be part of a composition. In their 2007 contribution, the authors did some fine tuning in code level and re-implemented the system in C++ (2006 it was written in C#) in order to improve the performance [60].

9.1.3 Zhang, Yu, et al.

In their composition system Zhang, Yu, et al. utilize hash tables that map the service to their output and to their input parameters. Using these tables, they can perform backward and forward service composition based on breadth-first search. A backwards search (like in our own approach) successively finds services that provide unknown parameters until all wanted outputs are satisfied. Forward searching in this context means that services that can be invoked with the known parameters are iteratively added to the composition until all wanted outputs can be generated. Zhang, Yue, et al. outline that backward searching outperforms forward search and thus, adopt it in their final solution. However, from the same department another contribution using forward search has also been handed in [61]. Another similarity to our system is that they also prefer SAX-based input parsing since it is more efficient than processing trees [62]. In 2007, Zhang, Raman, et al. extended their approach by using heuristic functions based on the number of branches and parameters of service compositions [63].

9.2 Interfacing Multi-Purpose Systems

We initially created our composition system for the challenge and later found it efficient and variable enough to integrate it into real systems like the ADDO project. Other contestants did this the other way round, i. e. modified an existing multi-purpose composition system for the contest. Such applications can thus be applied more broadly than specialized solutions, but on the other hand, trade in some performance.

9.2.1 Using USDL

The composition engine of Kona, Ajay, et al. for instance is based on the *Universal Service-Semantics Description Language* (USDL). In order to take part in the competition, they transformed the files in the service repository and the composition requests to USDL. After the transformation, a composer

¹⁴SWSDS = SEWSIP Web Service Discovery System, SEWIP = Semantic Web Services Integration Platform

written in Prolog could process and solve them [64]. In their 2007 contribution [65], the authors use a first-order logic approach based on Constraint Logic Programming over finite domains (CLP(FD)).

9.2.2 Interfacing with MOVE

In [66], another contribution interfacing with the *MOVE* framework for design and execution of business processes in virtual enterprises is discussed. This solution is interesting because it allows the service repositories to be managed by different handlers. It is now possible to load small sets of services into the repository instantly while reading larger sets on demand. Its composer is based on a planning system.

9.2.3 Interfacing with jUDDI+

jUDDI+ is an extended version of the open source UDDI [67] implementation by the Apache Software Foundation. jUDDI+ does not focus on pure semantic matching, but is also able to cope with non-exact matching and effects duplication. Therefore, it uses the description logics based reasoner MaMaS-tng¹⁵. In order to apply this approach in the WSC, a mapping to the internal data formats was required. [68]

9.2.4 The SCE

The *Multiagent Web Service Composition Engine* (SCE) [69] consist of two primary architectural components: the Java Agent Development Framework¹⁶ (JADE) and a service description repository. Here, services as well as composition requests are represented by agents. These agents communicate with each other and solve the requests cooperatively.

9.2.5 Interfacing WSPR

The *Web Service Planner* (WSPR) offers a highly specified forward composition algorithm. It is accessed via JSP and web services running on a web application server [70]. Such sophisticated interfaces are beneficial in real-world applications, in the context of the purely performance-related contest they may bias the results negatively.

10 Conclusions

In order to solve the 2006 and 2007 Web Service Challenges we utilized three different approaches, an uninformed search, an informed search, and a genetic algorithm. The uninformed search proofed generally unfeasible for large

¹⁵<http://sisinflab.poliba.it/MAMAS-tng/> [accessed 2007-10-25]

¹⁶<http://jade.tilab.com/> [accessed 2007-10-25]

service repositories. It can only provide a good performance if the resulting compositions are very short.

However, in the domain of web service composition, the maximum number of services in a composition is only limited by the number of services in the repositories and cannot be approximated by any heuristic. Therefore, any heuristic or meta-heuristic search cannot be better than the uninformed search in the case that a request is sent to the composer which cannot be satisfied. This is one reason why the uninformed approach was kept in our system, along with its reliability for short compositions.

Superior performance for all test sets could be obtained by utilizing problem-specific information encapsulated in a fine-tuned heuristic function to guide a greedy search. This approach is more efficient than the other two tested variants by a magnitude.

Genetic algorithms are much slower, but were also always able to provide correct results to all requests. To put it simple, the problem of semantic composition as defined in the context of the WSC is not complicated enough to fully unleash the potential of genetic algorithms. They cannot cope with the highly efficient heuristic used in the greedy search. We anticipate however, that, especially in practical applications, additional requirements will be imposed onto a service composition engine. Such requirements could include quality of service (QoS), the question for optimal parallelization, or the generation of complete BPEL [71] processes. In this case, heuristic search will most probably become insufficient but genetic algorithms and genetic programming [72, 23] will still be able to deliver good results.

In this report, we have discussed semantic composition in general way. The algorithms introduced here are not limited to semantic web service composition. Other applications, like the composition of program modules are also interesting. From general specifications what functionality is needed, a compiler could (in certain limits, of course) deduce the correct modules and code to be linked, using the same methods we use for building service processes.

List of Figures

1	The logo of the Web Service Challenge.	3
2	The basic cycle of evolutionary algorithms.	16
3	Bit-toggling mutation of string chromosomes.	18
4	Mutation of variable-length string chromosomes.	18
5	Crossover (recombination) of variable-length string chromosomes.	19
6	A sketch of the Pareto front in the genetic composition algorithm.	20
7	The WSC 2007 Composition System of Bleul and Weise.	23
8	The Knowledge Base and Service Registry of our Composition System.	24

List of Tables

1	Experimental results for the web service composers.	22
---	---	----

List of Algorithms

1	$S = dl_dfs(r, d)$	9
2	$S = iddfs(r)$	10
3	$S = webServiceCompositionIDDFS(R)$	11
4	$S = greedySearch(r)$	13
5	$r = c_{wsc}(S_1, S_2)$	14

References

- [1] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Pearson Education, Prentice Hall, second edition, ISBN: 9780133056990, September 19, 2002.
- [2] Jabir and J. W. Moore. A search for fundamental principles of software engineering. *Computer Standards & Interfaces*, 19:155–160, March 1998. 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].
- [3] Ingrid Wetzel. Information systems development with anticipation of change: Focussing on professional bureaucracies. In *Proceedings of Hawaii International Conference on Systems Sciences, HICSS 34*, Maui, Hawaii, USA, January 2001. IEEE Computer Society. Online available at <http://citeseer.ist.psu.edu/532081.html> and <http://swt-www.informatik.uni-hamburg.de/publications/download.php?id=177> [accessed 2007-09-02].
- [4] Eric A. Marks and Michael Bell. *Executive’s Guide to Service oriented architecture (SOA): A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons, Inc., Hoboken, NJ, ISBN: 978-0-470-03614-3, April 2006.
- [5] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, ISBN: 978-0131858589, August 2, 2005.
- [6] David Booth and Canyang Kevin Liu. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. World Wide Web Consortium (W3C), June 26, 2007. W3C Recommendation. Online available at <http://www.w3.org/TR/2007/REC-wsd120-primer-20070626> [accessed 2007-09-02].

- [7] 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].
- [8] 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 [9].
- [9] 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*, 1:77–106, 2005. See also <http://www.wsmo.org/> [accessed 2007-09-02] and [8].
- [10] Steffen Bleul and Thomas Weise. An ontology for quality-aware service discovery. In C. Zircins, G. Ortiz, W. Lamerdorf, , and W. Emmerich, editors, *Engineering Service Compositions: First International Workshop, WESC05*, Vrije Universiteit Amsterdam, The Netherlands, December 12, 2005, volume RC23821 of *IBM Research Report*. Yorktown Heights: IBM Research Division. See also [11]. Online available at <http://www.it-weise.de/documents/files/BW2005QASD.pdf> [accessed 2007-11-20].
- [11] Steffen Bleul, Thomas Weise, and Kurt Geihs. An ontology for quality-aware service discovery. *Computer Systems Science Engineering*, 21(4), July 2006. See also [10]. Special issue on “Engineering Design and Composition of Service-Oriented Applications”. Online available at <http://www.it-weise.de/documents/files/BWG2006QASD.pdf> [accessed 2007-11-20].
- [12] Steffen Bleul and Kurt Geihs. Addo: Automatic service brokering in service oriented architectures, project homepage. Online available at <http://www.vs.uni-kassel.de/ADD0/> [accessed 2007-09-02].
- [13] LSDIS Lab (Large Scale Distributed Information Systems), Department of Computer Science, University of Georgia. *METEOR-S: Semantic Web Services and Processes*, 2004. Online available at <http://lsdis.cs.uga.edu/projects/meteor-s/> [accessed 2007-09-02].
- [14] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. Ode sws: A framework for designing and composing semantic web services. *IEEE Intelligent Systems*, 19:24–31, July-August 2004. Online available at <http://iswc2004.semanticweb.org/demos/14/paper.pdf> [accessed 2007-09-02]. See also [15].

- [15] 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*, History Corner, main quad (Building 200), Stanford University, CA, USA, March 22-24, 2004. Online available at <http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf> [accessed 2007-09-02]. See also [14].
- [16] M. Brian Blake, Kwok Ching Tsui, and Andreas Wombacher. The eee-05 challenge: a new web service discovery and composition competition. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce, and e-Service, EEE'05*, March 29-April 1, 2005, pages 780-783. Online available at <http://ws-challenge.georgetown.edu/ws-challenge/The%20EEE.htm> [accessed 2007-09-02].
- [17] M. Brian Blake, William K.W. Cheung, Michael C. Jaeger, and Andreas Wombacher. Wsc-06: The web service challenge. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 505-508. See proceedings [73].
- [18] M. Brian Blake, William K.W. Cheung, Michael C. Jaeger, and Andreas Wombacher. Wsc-07: Evolving the web service challenge. In *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)*, 2006, pages 422-423. See proceedings [74].
- [19] Steffen Bleul, Thomas Weise, and Kurt Geihs. Large-scale service composition in semantic service discovery. In *Ws-Challenge Part: M. Brian Blake, Andreas Wombacher, Michel C. Jaeger, and William K. Cheung, editors, Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 427-429. See proceedings [73]. 1st place in 2006 WSC. Online available at [BWG2006WSC.pdf](#) [accessed 2007-11-20]. See 2007 WSC [20].
- [20] Steffen Bleul, Thomas Weise, and Kurt Geihs. Making a fast semantic service composition system faster. In *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)*, 2007, pages 517-520. See proceedings [74]. 2nd place in 2007 WSC. Online available at <http://www.it-weise.de/documents/files/BWG2007WSC.pdf> [accessed 2007-11-20]. See 2006 WSC [19].
- [21] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition*. World Wide Web Consortium (W3C), second edition, October 28, 2004. W3C Recommendation. Online available at <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/> [accessed 2007-09-02].

- [22] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. World Wide Web Consortium (W3C), September 29, 2007. W3C Recommendation. Online available at <http://www.w3.org/TR/2006/REC-xml-20060816> [accessed 2007-09-02].
- [23] Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Thomas Weise, 2007-11-20 edition, 2007. Online available at <http://www.it-weise.de/> [accessed 2007-11-20].
- [24] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, ISBN: 0137903952, December 2002.
- [25] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science. The MIT Press and McGraw-Hill, second edition, ISBN: 0-2625-3196-8, first edition: 978-0262031417, August 2001. First edition June 1990.
- [26] Werner Dilger. *Einführung in die Künstliche Intelligenz*. Chemnitz University of Technology, Faculty of Computer Science, Chair of Artificial Intelligence (Künstliche Intelligenz), April 2006. Lecture notes for the lectures artificial intelligence. Online available at <http://www.tu-chemnitz.de/informatik/KI/skripte.php> [accessed 2007-08-06].
- [27] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, second, revised and extended edition, ISBN: 978-3540224945, December 2004.
- [28] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors. *Modern Heuristic Search Methods*. Wiley, ISBN: 978-0471962809, December 1996.
- [29] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley series in artificial intelligence. Addison-Wesley Pub (Sd), ISBN: 978-0201055948, April 1984.
- [30] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, ISBN: 0195099710, January 1996.
- [31] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Computational Intelligence Library. Oxford University Press in cooperation with the Institute of Physics Publishing, Bristol, New York, ringbound edition, ISBN: 0750303921, April 1997.

- [32] Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1:3–17, April 1997. Online available at <http://sci2s.ugr.es/docencia/doctobio/EC-History-IEEEEC-1-1-1997.pdf> and <http://citeseer.ist.psu.edu/601414.html> [accessed 2007-08-24].
- [33] Carlos Artemio Coello Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, August 1999. Online available at <http://www.lania.mx/~ccoello/EM00/informationfinal.ps.gz> and <http://citeseer.ist.psu.edu/coello98comprehensive.html> [accessed 2007-08-25].
- [34] Carlos Artemio Coello Coello. An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In *1999 Congress on Evolutionary Computation*, 1999, pages 3–13. See proceedings [75]. Online available at <http://citeseer.ist.psu.edu/coellocoello99updated.html> [accessed 2007-08-25].
- [35] David A. Van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Air Force Institute of Technology, Air University, Wright-Patterson Air Force Base, Ohio, May 1999. AFIT/DS/ENG/99-01. Online available at <http://handle.dtic.mil/100.2/ADA364478> and <http://citeseer.ist.psu.edu/vanveldhuizen99multiobjective.html> [accessed 2007-08-17].
- [36] Kalyanmoy Deb. Evolutionary algorithms for multi-criterion optimization in engineering design. In Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 135–161. John Wiley & Sons, Ltd, Chichester, UK, 1999. Online available at <http://citeseer.ist.psu.edu/deb99evolutionary.html> and <http://www.lania.mx/~ccoello/EM00/deb99.ps.gz> [accessed 2007-08-25].
- [37] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. John Wiley & Sons, Inc., New York, NY, USA, ISBN: 978-0471873396, May 2001.
- [38] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part i: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 28(1):26–37, 1998. Online available at <http://citeseer.ist.psu.edu/fonseca98multiobjective.html> [accessed 2007-07-29]. See also [76].
- [39] Sanaz Mostaghim. *Multi-objective Evolutionary Algorithms: Data structures, Convergence and, Diversity*. PhD thesis, Fakultät für

- Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Deutschland (Germany), 2004. Online available at <http://deposit.ddb.de/cgi-bin/dokserv?idn=974405604> and <http://ubdata.uni-paderborn.de/ediss/14/2004/mostaghi/disserta.pdf> [accessed 2007-08-17].
- [40] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, ISBN: 0-2626-5040-1, July 1994.
- [41] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, ISBN: 0-2620-6141-4, 978-0-262-06141-4, August 1991.
- [42] Vira Chankong and Yacov Y. Haimes. *Multiobjective Decision Making Theory and Methodology*. North-Holland, Elsevier, Dover Publications, New York, ISBN: 978-0444007100, 978-0486462899, January 1983.
- [43] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. Krieger Pub Co, reprint edition, ISBN: 978-0894643934, August 1989.
- [44] Yacov Y. Haimes and Ralph E. Steuer, editors. *Proceedings of the 14th International Conference on Multiple Criteria Decision Making: Research and Practice in Multi Criteria Decision Making (MCDM'1998)*, University of Virginia, Charlottesville, Virginia, USA, June 12-18, 1998, volume 487 of *Lecture Notes in Economics and Mathematical Systems*. Springer, ISBN: 978-3540672661. See <http://www.virginia.edu/~risk/mcdm98.html> [accessed 2007-09-10]. Published June 15, 2000.
- [45] E. A. Galperin. Pareto analysis vis-à-vis balance space approach in multiobjective global optimization. *Journal of Optimization Theory and Applications*, 93(3):533–545, June 1997. Online available at <http://www.springerlink.com/content/m71638106736h581/fulltext.pdf> and <http://dx.doi.org/10.1023/A:1022639028824> [accessed 2007-09-10].
- [46] Aharon Ben-Tal. Characterization of pareto and lexicographic optimal solutions. In *Proceedings of the Third Conference on Multiple Criteria Decision Making: Theory and Application*, 1979, pages 1–11. See proceedings [77].
- [47] John Henry Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962. Online available at <http://portal.acm.org/citation.cfm?id=321128> [accessed 2007-07-28].
- [48] John Henry Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, ISBN: 978-0262581111, 1975. Reprinted by MIT Press, April 1992.
- [49] Jack L. Crosby. *Computer Simulation in Genetics*. John Wiley and Sons Ltd, ISBN: 0-4711-8880-8, January 1973.

- [50] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, first edition, ISBN: 0201157675, January 1989.
- [51] Jörg Heitkötter and David Beasley, editors. *Hitch-Hiker's Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ)*. ENCORE (The Evolutionary Computation REpository Network), 1998. USENET: comp.ai.genetic. Online available at <http://www.cse.dmu.ac.uk/~rij/gafaq/top.htm> and <http://alife.santafe.edu/~joke/encore/www/> [accessed 2007-07-03].
- [52] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995. Online available at <http://citeseer.ist.psu.edu/108172.html> [accessed 2007-07-29].
- [53] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. In *Practical Approaches to Multi-Objective Optimization*, 2004. See proceedings [78]. Online available at <http://drops.dagstuhl.de/opus/volltexte/2005/237/> [accessed 2007-09-19].
- [54] Carlos M. Fonseca. Decision making in evolutionary optimization (abstract of invited talk). In *4th International Conference on Evolutionary Multi-Criterion Optimization*, 2007. See proceedings [79].
- [55] Carlos M. Fonseca. Preference articulation in evolutionary multiobjective optimisation – plenary talk. In *Proceedings of the Seventh International Conference on Hybrid Intelligent Systems, HIS 2007*, 2007. See proceedings [80].
- [56] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, May 2003. Online available at <http://www.ibm.com/developerworks/library/specification/ws-bpel/> [accessed 2007-09-03].
- [57] Marco Aiello, Christian Platzer, Florian Rosenberg, Huy Tran, Martin Vasko, and Schahram Dustdar. Web service indexing for efficient retrieval and composition. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 424–426. See proceedings [73]. Online available at <https://berlin.vitalab.tuwien.ac.at/~florian/papers/cec2006.pdf> [accessed 2007-10-25].
- [58] Lukasz Juszcyk, Anton Michlmayer, and Christian Platzer. Large scale web service discovery and composition using high performance in-memory indexing. In *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)*, 2007, pages 509–512. See

- proceedings [74]. Online available at <https://berlin.vitalab.tuwien.ac.at/~florian/papers/cec2007.pdf> [accessed 2007-10-24].
- [59] Bin Xu, Tao Li, Zhifeng Gu, and Gang Wu. SWSDS: Quick web service discovery and composition in SEWSIP. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 429–451. See proceedings [73].
 - [60] Zhifeng Gu, Bin Xu, and Juanzi Li. Inheritance-aware document-driven service composition. In *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)*, 2007, pages 513–516. See proceedings [74].
 - [61] Mohammad A. Makhzan and Kwei-Jay Lin. Solution to a complete web service discovery and composition. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 455–457. See proceedings [73].
 - [62] Yue Zhang, Tao Yu, Krishna Raman, and Kwei-Jay Lin. Strategies for efficient syntactical and semantic web services discovery and composition. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 452–454. See proceedings [73].
 - [63] Yue Zhang, Krishna Raman, Mark Panahi, and Kwei-Jay Lin. Heuristic-based service composition for business processes with branching and merging. In *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)*, 2007, pages 525–528. See proceedings [74].
 - [64] Srividya Kona, Ajay Bansal, Gopal Gupta, and Thomas D. Hite. Web service discovery and compositing using usdl. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 430–432. See proceedings [73].
 - [65] Srividya Kona, Ajay Bansal, Gopal Gupta, and Thomas D. Hite. Semantics-based web service composition engine. In *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)*, 2007, pages 521–524. See proceedings [74]. Online available at <http://www.utd.edu/~sxk038200/research/wsc07.pdf> [accessed 2007-10-25].

- [66] Jürgen Dorn, Albert Rainer, and Peter Hrastnik. Toward semantic composition of web services with MOVE. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 437–438. See proceedings [73]. Online available at <http://move.ec3.at/Papers/WSC06.pdf> [accessed 2007-10-25].
- [67] Luc Clement, Andrew Hatley, Claus von Riegen, and Tony Rogers. *UDDI Version 3.0.2 – UDDI Spec Technical Committee Draft, Dated 20041019*. Organization for the Advancement of Structured Information Standards (OASIS), October 19, 2004. Online available at <http://www.oasis-open.org/committees/uddi-spec/doc/spec/v3/uddi-v3.0.2-20041019.htm> [accessed 2007-10-25].
- [68] Francesco Colasuonno, Sefano Coppi, Azzurra Ragone, and Luca L. Scordia. jUDDI+: A semantic web services registry enabling semantic discovery and composition. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, 2006, pages 442–444. See proceedings [73].
- [69] Paul A. Buhler, Dominic Greenwood, and George Weichhart. A multiagent web service composition engine, revisited. In *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)*, 2007, pages 529–532. See proceedings [74].
- [70] Seog-Chan Oh, Jung-Woon Yoo, Hyunyoung Kil, Dongwon Lee, and Soundar R. T. Kumara. Semantic web-service discovery and composition using flexible parameter matching. In *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)*, 2007, pages 533–536. See proceedings [74].
- [71] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. *Web Services Business Process Execution Language Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS), April 11, 2007. Online available at <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf> [accessed 2007-10-25]. Technical Committee: OASIS Web Services Business Process Execution Language (WSBPEL) TC.
- [72] John R. Koza. *Genetic Programming, On the Programming of Computers by Means of Natural Selection*. A Bradford Book, The MIT Press,

Cambridge, Massachusetts, 1992 first edition, 1993 second edition, ISBN: 0262111705, 1992.

- [73] *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, The Westin San Francisco Airport, 1 Old Bayshore Highway, Millbrae, United States, June 26-29, 2006. IEEE Computer Society, Los Alamitos, California, Washington, Tokyo, ISBN: 978-0-7695-2511-2.
- [74] IEEE Computer Society. *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)*, National Center of Sciences, Tokyo, Japan, July 23-26, 2007. IEEE Computer Society, ISBN: 978-0-7695-2913-4.
- [75] Peter John Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC99*, Mayflower Hotel, Washington D.C., USA, July 6-9, 1999, volume 1-3, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-5536-9, 0-7803-5537-7. CEC-99 - A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE. Library of Congress Number: 99-61143.
- [76] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part ii: Application example. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 28(1):38–47, 1998. Online available at <http://citeseer.ist.psu.edu/27937.html> [accessed 2007-09-19]. See also [38].
- [77] Joel N. Morse, editor. *Proceedings of the 4th International Conference on Multiple Criteria Decision Making: Organizations, Multiple Agents With Multiple Criteria (MCDM'1980)*, Newark, Delaware, USA, 1980, Lecture Notes in Economics and Mathematical Systems. Springer, ISBN: 978-0387108216. Published in July 1981.
- [78] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Ralph E. Steuer, editors. *Practical Approaches to Multi-Objective Optimization*, Dagstuhl, Germany, November 7-12, 2004, number 04461 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany IBFI, ISSN: 1862-4405. Published in 2005. Online available at <http://drops.dagstuhl.de/portals/index.php?semnr=04461> and <http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=04461> [accessed 2007-09-19].
- [79] Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata, editors. *Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization (EMO'2007)*, Matsushima/Sendai, Japan, March 5-8, 2007, volume 4403/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, ISBN:

978-3-540-70927-5, ISSN: 0302-9743 (Print) 1611-3349 (Online). See <http://www.is.doshisha.ac.jp/emo2007/> [accessed 2007-09-11].

- [80] Andreas König, Mario Köppen, Ajith Abraham, Christian Igel, and Nikola Kasabov, editors. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, Fraunhofer-Center, University of Kaiserslautern, Kaiserslautern, Germany, September 17-19, 2007. IEEE Computer Society, ISBN: 0-7695-2946-1. Library of Congress Number 2007936727, Product Number E2946. see <http://his07.hybridsystem.com/> [accessed 2007-09-01].

Index

- ADDO, 23, 26
- Architecture
 - service oriented, 2
- Best-First Search, 12
- BPEL, 28
- BPEL4WS, 24
- Contravariance, 4
- Covariance, 5
- Depth-First Search
 - iterative deepening, 9
- Depth-limited Search, 8
- DFS, 8
- DOM, 26
- Domination, 17
- EMOO, 17
- Evolutionary Algorithm
 - multi-objective, 17
- expand, 8
- Exploration, 8
- Greedy Search, 12
- IDDFS, 9
- isGoal, 8
- JADE, 27
- jUDDI+, 27
- MOEA, 16, 17
- MOVE, 27
- Multi-objective, 16
- Mutation, 18
- Order
 - partial, 17
- OWL-S, 2
- Pareto, 17
 - optimal, 17
- Partial order, 17
- QoS, 28
- Quality of Service, 28
- SAX, 25, 26
- SCE, 27
- Search
 - best-first, 12
 - depth-first
 - iterative deepening, 9
 - depth-limited, 8
 - greedy, 12
 - uninformed, 8
- Service Oriented Architecture, 2
- SOA, 2
- StAX, 25
- SWSDS, 26
- Uninformed Search, 8
- USDL, 26
- VitaLab, 25
- Web Service Challenge, 3
- WS-Challenge, 3
- WSC, 1, 3, 25, 28
- WSDL, 2, 6
- WSPR, 27
- XML, 25
- XSD, 6

```

@techreport{WBG2007WSCb,
author
= {Thomas Weise and Steffen Bleul and Kurt Geihs},
title
= {Web Service Composition Systems for the Web Service Challenge - A
Detailed Review},
series
= {Kasseler Informatikschriften (KIS)},
number
= {2007, 7},
pages
= {1--40},
publisher
= {University of Kassel},
year
= {2007},
month
= nov # {-19},
type
= {Kasseler Informatikschriften (KIS)},
location
= {University of Kassel},
address
= {University of Kassel},
institution
= {University of Kassel},
organization
= {University of Kassel},
school
= {University of Kassel},
note
= {A technical report describing our experiences from the 2006 and 2007
Web Service Challenges. Persistent Identifier:
urn:nbn:de:hebis:34-2007111919638\\
The work is online available at
http://www.it-weise.de/documents/index.html\#WBG2007WSCb. \\
The publication can be downloaded at
http://www.it-weise.de/documents/files/WBG2007WSCb.pdf. \\
Contact Thomas Weise at tweise@gmx.de or http://www.it-weise.de/. },
copyright
= {unrestricted},
abstract
= {This report gives a detailed discussion on the system, algorithms, and
techniques that we have applied in order to solve the Web Service
Challenges (WSC) of the years 2006 and 2007. These international
contests are focused on semantic web service composition. In each
challenge of the contests, a repository of web services is given. The
input and output parameters of the services in the repository are
annotated with semantic concepts. A query to a semantic composition
engine contains a set of available input concepts and a set of wanted
output concepts. In order to employ an offered service for a requested
role, the concepts of the input parameters of the offered operations
must be more general than requested (contravariance). In contrast, the
concepts of the output parameters of the offered service must be more
specific than requested (covariance). The engine should respond to a
query by providing a valid composition as fast as possible. We discuss
three different methods for web service composition: an uninformed
search in form of an IDDFS algorithm, a greedy informed search based on
heuristic functions, and a multi-objective genetic algorithm.},
contents
= { * Introduction\\
* Semantic Service Composition\\
* The Problem Definition\\
* An (Uninformed) Algorithm based on IDDFS\\
* An (Informed) Heuristic Approach\\
* A Genetic Approach\\
* Experimental Results\\
* Architectural Considerations\\
* Related Work\\
* Conclusions},
keywords
= {Web Service Composition, Web Service Matching, Semantic, Semantic
Matching, Uninformed Search, Web Service Challenge, WSC, Semantic
Compositions, Semantic Web, IDDFS, Iterative Deepening Depth-First
Search, Greedy Search, Genetic Algorithm, Best-First Search},
language
= {en},
price
= {40},
url
= {http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007111919638}
}

```