

Different Approaches to Semantic Web Service Composition

Thomas Weise, Steffen Bleul, Diana Comes, and Kurt Geihs

Distributed Systems Group

University of Kassel

Wilhelmshöher Allee 73

34121 Kassel

Email: {weise|bleul|comes|geihs}@vs.uni-kassel.de

Preview

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

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

Abstract—Semantic web service composition is about finding services from a repository that are able to accomplish a specified task if executed. The task is defined in a form of a composition request which contains a set of available input parameters and a set of wanted output parameters. Instead of the parameter values, concepts from an ontology describing their semantics are passed to the composition engine. The parameters of the services in the repository the composer works on are semantically annotated in the same way as the parameters in the request. The composer then finds a sequence of services, called a composition. If the input parameters given in the request are provided, the services of this sequence can subsequently be executed and will finally produce the wanted output parameters.

In this paper, three different approaches to semantic web service composition are formally defined and compared with each other: an uninformed search in form of an IDDFS algorithm, a greedy informed search based on heuristic functions, and a multi-objective genetic algorithm.

I. INTRODUCTION

The necessity for fast service composition systems 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 like accounting, billing, the workflows, and even in the office software. If external vendors, suppliers, or customers change, the interfaces to their IT systems must be newly created or modified too. Hence, the architecture of corporate software has to be built with the anticipation of changes and updates [1, 2].

A SOA is the ideal architecture for such systems [3, 4]. Service oriented architectures allow us to modularize the business logic and to implement it in the form of services accessible in a network. Services are building blocks for service processes which represent the workflows of an enterprise. They can be added, removed, and updated at runtime without interfering

with the ongoing business. A SOA can be seen as a complex system with manifold services as well as $n:m$ dependencies between services and applications:

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

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

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 [5] specify the order and types of service parameters and return values. Semantic interface description languages like OWL-S [6] or WSMO [7, 8] annotate these parameters with a meaning. While WSDL can be used to define a parameter `myisbn` of the type `String`, with OWL-S we can define that `myisbn` expects a `String` which actually contains an `ISBN`. Via a taxonomy we can now deduce that values which are annotated as either `ISBN-10` or `ISBN-13`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 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 semantic service discovery [9, 10, 11, 12]. Still, there is a substantial lack of research on algorithms and system design for fast response service discovery. This is especially the case in ser-

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].

vice 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.

II. 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} . All concepts that exist in the knowledge base are members of \mathbb{M} and can be represented as nodes in a wood of taxonomy trees.

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

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

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

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

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

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

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

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

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

III. UNINFORMED APPROACHES – IDDFS

The most general and straightforward approach to web service composition is the uninformed search. Uninformed search algorithms do not make use of any information different from goal predicates as defined in 1. We can build such a composition algorithm based on 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.

Algorithm 1: $S = IDDFSComposition(R)$

Input: R the composition request

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

Data: in, out current parameter sets

Output: S a valid service composition solving R

```

1 begin
2   maxDepth ← 2
3   repeat
4     S ← dl_dfs(R.in, R.out, ∅, 1)
5     maxDepth ← maxDepth + 1
6   until S ≠ ∅
7 end
8 dl_dfs(in, out, composition, depth)
9 begin
10  foreach A ∈ out do
11    foreach s ∈ promising(A) do
12      wanted ← out
13      foreach B ∈ wanted do
14        if ∃ C ∈ s.out : subsumes(B, C) then
15          wanted ← wanted \ {B}
16        foreach D ∈ s.in do
17          if ∄ E ∈ in : subsumes(D, E) then
18            wanted ← wanted ∪ {D}
19          comp ← s ⊕ composition
20          if wanted = ∅ then
21            return comp
22          else
23            if depth < maxDepth then comp ←
24              dl_dfs(in, wanted, comp, depth + 1)
25            if comp ≠ ∅ then return comp
26        return ∅
27 end

```

1 builds a valid web service composition starting from the back. In each recursion, its internal helper method `dl_dfs` tests all elements A of the set *wanted* of yet unknown parameters. It then iterates over the set of 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` 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 1 iteratively invokes `dl_dfs` by increasing the depth limit step by step, until a valid solution is found.

IV. AN (INFORMED) HEURISTIC APPROACH

The IDDFS algorithm just discussed 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. In an informed search, a heuristic c helps to decide which nodes are to be expanded next. If the heuristic is good, such algorithms may dramatically outperform uninformed strategies [13, 14]. As second composition method we therefore define a greedy algorithm that internally sorts the list of currently known candidate compositions in *descending* order according to a heuristic in form of a comparator function $c : \mathbb{S}^n \in \mathbb{R}$. The comparator function $c(S_1, S_2)$ will be below zero if S_1 seems to be closer to the solution than S_2 and greater than zero if S_2 is a more prospective candidate. Thus, the best elements will be at the end of the list X in 2.

Algorithm 2: $S = \text{greedyComposition}(R, c)$

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

```

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

We can easily derive different comparator functions for web service composition. In our experiments and real-world experiences, the function c_{cmp} has proven to be most efficient. It combines the size of the set unsatisfied parameters $\forall A \in \text{wanted}(S) \Rightarrow \exists s \in S : A \in s.in \wedge A \notin \text{known}(S)$, the composition lengths, the number of satisfied parameters $\forall B \in \text{eliminated}(S) \Rightarrow \exists s \in S : B \in s.in \wedge B \in \text{known}(S)$, and the number of known concepts $\text{known}(S) = R.in \cup_{s \in S} s.out$ as defined in 3.

Algorithm 3: $r = c_{cmp}(S_1, S_2)$

Input: S_1, S_2 two composition candidates

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

```

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

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

V. A GENETIC APPROACH

Evolutionary algorithms (EAs) [15, 14] are generic, population-based meta-heuristic optimization algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection and survival of the fittest. The advantage of evolutionary algorithms compared to other optimization methods is that they make only few assumptions about the underlying fitness landscape and therefore perform consistently well in many different problem categories.

All evolutionary algorithms proceed in principle according to the scheme illustrated in 1:

- 1) Initially, a population of totally random individuals is created.
- 2) All of them are tested for their utility as solution.
- 3) Based on this evaluation, fitness values are assigned to 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 solution candidates.

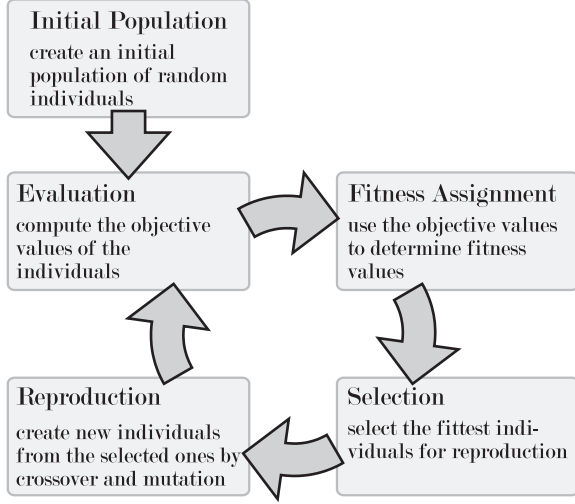


Fig. 1: The basic cycle of evolutionary algorithms.

- 6) If the termination criterion is met, the evolution stops here. Otherwise, it continues at step 2.

For the reproduction of solution candidates, EAs employ two different operators:

- Mutation slightly modifies one existing individual and
- crossover combines two solution candidate, creating offspring with features of both.

A. An Evolutionary Composition Algorithm

In order to use a evolutionary 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. Then, 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 determine whether the search should prefer growing or shrinking the solution candidates.

$$mutate_1(S) \equiv \begin{cases} \{s_2, s_3, \dots, s_{|S|}\} & \text{if } |S| > 1 \\ S & \text{otherwise} \end{cases} \quad (3)$$

$$mutate_2(S) \equiv s \oplus S : \begin{matrix} A \in wanted(S) \wedge \\ s \in promising(A) \end{matrix} \quad (4)$$

$$mutate(S) \equiv \begin{cases} mutate_1(S) & \text{if } random() > \sigma \\ mutate_2(S) & \text{otherwise} \end{cases} \quad (5)$$

A new operation for building the initial random configurations can be defined as a sequence of $mutate_2$ invocations of random length. Initially, $mutate_2(\emptyset)$ 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 create a new individual.

1) *The Comparator Function and Pareto Optimization:* As driving force for the evolutionary process we can reuse the comparator function c_{cmp} as specified as for the greedy search in 3 on the preceding page. 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 the multi-objective Pareto optimization technique. This has the drawback that it spreads the pressure of the optimization process over the complete Pareto frontier.

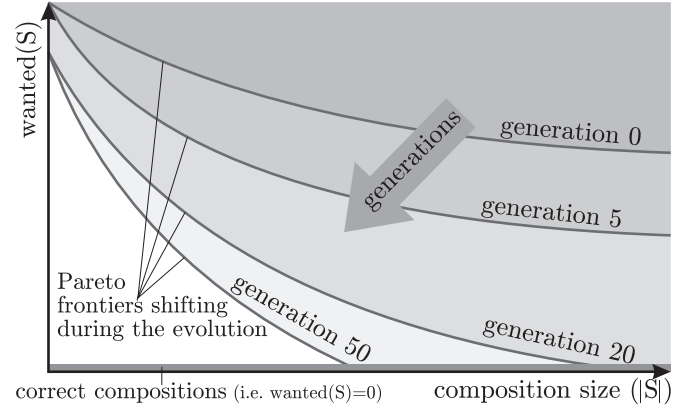


Fig. 2: A sketch of the Pareto front in the genetic composition algorithm.

2 visualizes the multi-objective optimization problem “web service composition” by sketching a characteristic example for Pareto frontiers of several generations of an evolutionary algorithm. In 2, we concentrate on the two objectives *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 size objective is thus to be minimized.

In the example, the first five or so generations are not able to produce good compositions yet. We 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

TABLE I: Experimental results for the web service composers.

Test	Size of Composi.	No. of Concepts	No. of Services	—IDDFS —	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

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 slows down the progress into the specific direction where $wanted(S)$ decreases.

The comparator function c_{cmp} has proven to be more efficient in focusing the evolution on this part of the search space. The evolution driven by this function is superior in performance and hence, is used in our experiments.

B. Experimental Results

In I we illustrate the mean processing times that the different algorithms introduced in this report when applied multiple times to various composition tasks. The test sets are based on the international Web Service Challenge [16, 17, 18] and more thoroughly discussed in [19]. They feature knowledge bases \mathbb{M} and service registries \mathbb{S} of different sizes, as well as a variety of composition lengths. We have repeated the experiments multiple times 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. On the downside, a heuristic may be misleading in some cases and could lead to a very long computation time in the worst case.

The genetic algorithm (population size 1024) was able to resolve all composition requests correctly for all knowledge bases 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.

VI. RELATED WORK

A. SWSDS

SWSDS² composition system [20, 21] can be used for both, syntactic and semantic matching by simply switching a so-called search index. SWSDS uses a composition algorithm similar to our uninformed IDDFS variant but extended with the constraint that each service is only considered one time to be part of a composition. This mitigates the weak performance of the uninformed search but trades in the guaranteed optimality of the result.

B. Zhang et al.

In their composition system [22], Zhang et al. utilize hash tables that map the service to their output and to their input parameters. Using these tables, both forward and backward service composition, can be performed 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 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 et al. made the experience that backward searching outperforms forward search. Since 2007, they apply heuristics based on the number of branches and parameters of service compositions [23].

C. USDL

The composition engine of Kona et al. is based on the *Universal Service-Semantics Description Language* (USDL). WSDL files in the service repository and the composition requests are transformed to USDL. After that, a composer written in Prolog can process and solve them [24]. In 2007, a first-order logic approach based on Constraint Logic Programming over finite domains (CLP(FD)) was used for solving the requests [25].

D. The SCE

The *Multiagent Web Service Composition Engine* (SCE) [26] consist of two primary architectural components: the Java Agent Development Framework³ (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. In general, AI-based approaches like USDL and SCE are slower than optimized problem specific methods like the greedy algorithm applied by us, as proved in the 2006/2007 Web Service Challenges.

²SWSDS = SEWSIP Web Service Discovery System, SEWIP = Semantic Web Services Integration Platform

³<http://jade.tilab.com/> [accessed 2007-10-25]

E. jUDDI+

jUDDI+ [27] is an extended version of the open source UDDI [28] 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-tng⁴. The connection to UDDI provides versatility but also brings overhead which slows down the composition system in direct comparisons to ours performed at the 2006/2007 WSCs.

VII. THE ADDO COMPOSITION SYSTEM

The composition algorithms discussed in this paper are utilized by the ADDOaction⁵ project, which allows business applications to perform semantic service discovery and composition. In [?], we presented the ADDOaction discovery and

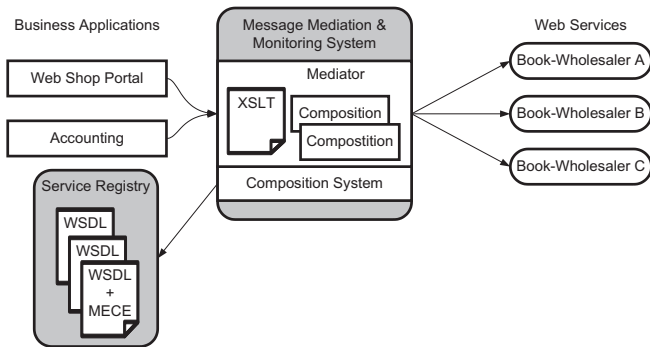


Fig. 3: Service Integration and Mediation.

mediation system for Web Service integration. The system uses WSDL descriptions enriched with the Mediation Contract Extension (MECE). It is able to automatically generate the XSLT [29] documents necessary to convert the arbitrarily complex message formats used for the data exchange between the services. Furthermore, it also automatically discovers and integrates plugins needed for the message exchange where XSLT does not suffice. XSLT is, for instance, not able to convert between different data types or units like different currencies. This can only be performed by additional software components which can be added as plug-ins into the system at runtime. The ADDOaction system integrates the necessary plug-ins into the compositions and the mediation system is now able to provide transparent access to the service compositions exactly as specified in the composition request by the client application.

An application accesses the composition system illustrated in 4 by submitting a service request through its *Web Service Interface* and also provides the service descriptions and their semantic annotations in form of WSDL and XSD formatted files or OWL-S descriptions. These documents are parsed by a fast *SAX-based Input Parser*. The composition process itself is started by the *Strategy Planer* which allocates the system

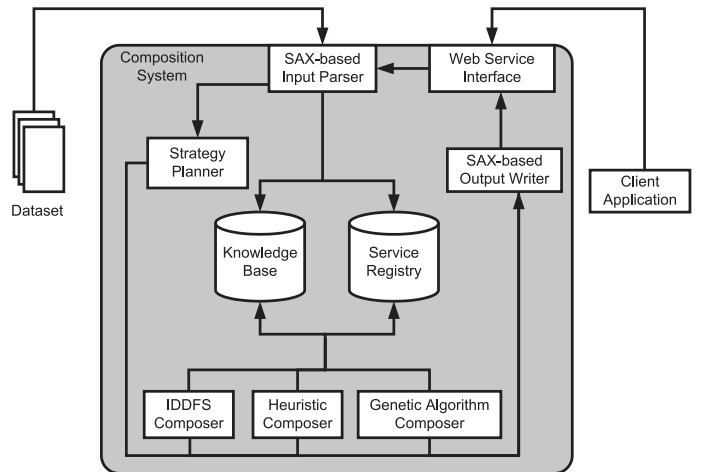


Fig. 4: The composition system.

resources and chooses the composition algorithm that seems to be most appropriate for the given problem.

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

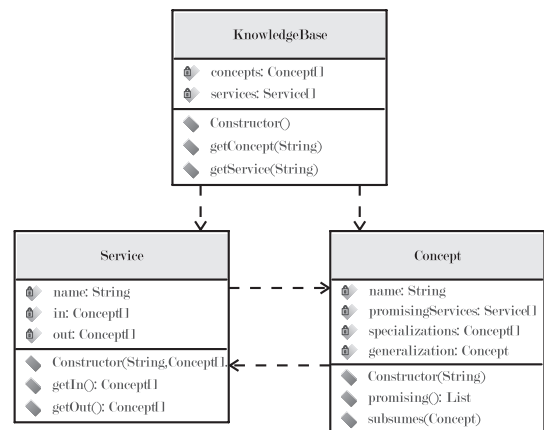


Fig. 5: The knowledge base.

The knowledge base and the service register are constituted by instances of the classes *Concept* and *Service*, as illustrated in 5. Instances of *Concept* provides the method *promising* which corresponds to the predicate *promising* introduced in 2. In order to determine its result, all the specializations of the concept have to be traversed and their promising services have to be accumulated. The crux of the routine is that this costly traversal is only performed once per concept. Our experiments have shown that the resource *memory* is not a bottleneck even for largest service repositories. Hence,

⁴<http://sisinflab.poliba.it/MAMAS-tng/> [accessed 2007-10-25]

⁵<http://www.vs.uni-kassel.de/ADDO/>

promising caches its results. This caching is done in a way that is thread-safe on one hand and does not need any synchronization on the other. The same holds for all features of the knowledge base: multiple algorithms may run in parallel, using the same knowledge base without any synchronization-induced delay, race conditions or inconsistencies.

VIII. CONCLUSIONS

In this research work, we have formally defined and evaluated three different approaches for web service composition: an uninformed search, an informed search, and an evolutionary algorithm. The uninformed search proved generally unfeasible for large service repositories. It can only provide a good performance if the resulting compositions are very short. However, in case that a request is sent to the composer which cannot be satisfied, even heuristic approaches have to test all valid service combinations and thus cannot be better than IDDFS.

In practical experiments based on the data sets of the Web Service Challenge, superior performance could be measured 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.

Evolutionary 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 [16] is not complicated enough to fully unleash the potential of EAs. Here, they cannot cope with the highly efficient heuristic used in the greedy search. We anticipate that, especially in practical applications, additional requirements will be imposed onto a service composition engine. Such requirements will include quality of service (QoS), the question for optimal parallelization, or the generation of complete BPEL [30] processes. In this case, heuristic search will most probably become insufficient but EAs will still be able to deliver good results.

REFERENCES

- [1] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Pearson Education, Prentice Hall, second edition, September 19, 2002. ISBN 9780133056990.
- [2] Jabir and J. W. Moore from the 7803 Whiterim Terrace, Potomac, MD 20854, USA. A search for fundamental principles of software engineering. *Computer Standards & Interfaces*, 19:155–160, March 1998. doi: 10.1016/S0920-5489(98)00009-9. Online available at [http://dx.doi.org/10.1016/S0920-5489\(98\)00009-9](http://dx.doi.org/10.1016/S0920-5489(98)00009-9) and <http://www.gelog.etsmtl.ca/publications/pdf/249.pdf> [accessed 2007-09-02].
- [3] 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, April 2006. ISBN 978-0-470-03614-3.
- [4] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, August 2, 2005. ISBN 978-0131858589.
- [5] 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-wsdl20-primer-20070626> [accessed 2007-09-02].
- [6] 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].
- [7] 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 [8].
- [8] 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 [7].
- [9] Steffen Bleul and Thomas Weise from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. An Ontology for Quality-Aware Service Discovery. In C. Zirpins, G. Ortiz, W. Lamerdorf, and W. Emmerich, editors, *Engineering Service Compositions: First International Workshop, WESC05*, volume RC23821 of *IBM Research Report*. Yorktown Heights: IBM Research Division, Dezember 12, 2005, Vrije Universiteit Amsterdam, The Netherlands. See also [11]. Online available at <http://www.it-weise.de/documents/files/BW2005QASD.pdf> [accessed 2008-06-17].
- [10] 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. ISSN 1541-1672. doi: 10.1109/MIS.2004.32. Online available at <http://iswc2004.semanticweb.org/demos/14/paper.pdf> [accessed 2007-09-02]. See also [12].
- [11] Steffen Bleul, Thomas Weise, and Kurt Geihs from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. An Ontology for Quality-Aware Service Discovery. *Com-*

- puter Systems Science Engineering*, 21(4), July 2006. See also [9]. Special issue on “Engineering Design and Composition of Service-Oriented Applications”. Online available at <http://www.it-weise.de/documents/files/BWG2006QASD.pdf> [accessed 2008-06-17].
- [12] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. A Framework for Designing and Composing Semantic Web Services. In *Semantic Web Services, First International Semantic Web Services Symposium, Proceedings of 2004 AAAI Spring Symposium Series*, March 22–24, 2004, History Corner, main quad (Building 200), Stanford University, CA, USA. Online available at <http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf> [accessed 2007-09-02]. See also [10].
- [13] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, second, revised and extended edition, Dezember 2004. ISBN 978-3540224945.
- [14] Thomas Weise from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. *Global Optimization Algorithms – Theory and Application*. Thomas Weise, second edition, 2008. Online available at <http://www.it-weise.de/> [accessed 2008-06-17].
- [15] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, January 1996. ISBN 0195099710.
- [16] 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)*, pages 422–423, 2006. In proceedings [18].
- [17] *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC’06) and Enterprise Computing, E-Commerce and E-Services (EEE’06)*, June 26–29, 2006, The Westin San Francisco Airport, 1 Old Bayshore Highway, Millbrae, United States. IEEE Computer Society, Los Alamitos, California, Washington, Tokyo. ISBN 978-0-7695-2511-2.
- [18] *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)*, July 23–26, 2007, National Center of Sciences, Tokyo, Japan. IEEE Computer Society, IEEE Computer Society. ISBN 978-0-7695-2913-4.
- [19] Thomas Weise, Steffen Bleul, and Kurt Geihs from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. Web Service Composition Systems for the Web Service Challenge – A Detailed Review. *Kasseler Informatikschriften (KIS) 2007, 7*, University of Kassel, FB16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, University of Kassel, November 19, 2007. Persistent Identifier: urn:nbn:de:hebis:34-2007111919638. Online available at <http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007111919638> and <http://www.it-weise.de/documents/files/WBG2007WSCb.pdf> [accessed 2007-11-20]. See also [31, 32].
- [20] 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)*, pages 429–451, 2006. In proceedings [17].
- [21] 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)*, pages 513–516, 2007. In proceedings [18].
- [22] 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)*, pages 452–454, 2006. In proceedings [17].
- [23] 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)*, pages 525–528, 2007. In proceedings [18].
- [24] 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)*, pages 430–432, 2006. In proceedings [17].
- [25] 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)*, pages 521–524, 2007. In proceedings [18]. Online available at <http://www.utd.edu/~sxk038200/research/wsc07.pdf> [accessed 2007-10-25].
- [26] 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)*, pages 529–532, 2007. In proceedings [18].
- [27] 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)*, pages 442–444, 2006. In proceedings [17].
- [28] Luc Clement, Andrew Hately, 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].
- [29] James Clark. *XSL Transformations (XSLT)*. World Wide Web Consortium (W3C), November 16, 1999. W3C Recommendation. Online available at <http://www.w3.org/TR/1999/REC-xslt-19991116> [accessed 2008-03-18].
- [30] 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.
- [31] Steffen Bleul, Thomas Weise, and Kurt Geihs from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. 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)*, pages 427–429, 2006. In proceedings [17]. 1st place in 2006 WSC. Online available at <http://www.it-weise.de/documents/files/BWG2006WSC.pdf> [accessed 2008-06-17]. See 2007 WSC [32] and [19].
- [32] Steffen Bleul, Thomas Weise, and Kurt Geihs from the University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany. 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)*, pages 517–520, 2007. In proceedings [18]. 2nd place in 2007 WSC. Online available at <http://www.it-weise.de/documents/files/BWG2007WSC.pdf> [accessed 2008-06-17]. See 2006 WSC [31] and [19].

```
@inproceedings{WBCG2008ICIW,  
  author      = {Thomas Weise and Steffen Bleul and Diana Comes and Kurt Geihs},  
  title       = {Different Approaches to Semantic Web Service Composition},  
  affiliation  = {Distributed Systems Group,  
                 FB 16 -- Elektrotechnik und Informatik,  
                 University of Kassel,  
                 Wilhelmsh{\"}her Allee 73,  
                 34121 Kassel, Germany},  
  booktitle   = {Proceedings of The Third International Conference on Internet  
                 and Web Applications and Services, ICIW 2008},  
  editor      = {Abdelhamid Mellouk and Jun Bi and Guadalupe Ortiz and  
                 Dickson K. W. Chiu and Manuela Popescu},  
  ISBN        = {978-0-7695-3163-2},  
  pages       = {90--96},  
  year        = {2008},  
  month       = jun # {~8--13,},  
  location    = {Athens, Greece},  
  publisher   = {IEEE Computer Society Press},  
  address     = {Los Alamitos, CA, USA},  
  url         = {http://www.it-weise.de/documents/files/WBCG2008ICIW.pdf},  
  note        = {Product Number: E3163. BMS Part Number: CFP0816C-CDR.  
                 Library of Congress Number: 2008922600},  
}
```