Scaling up Service Composition with Downloadable Directory Digests

Walter Binder, Ion Constantinescu, and Boi Faltings

Ecole Polytechnique Fédérale de Lausanne (EPFL)
Artificial Intelligence Laboratory
CH-1015 Lausanne, Switzerland
firstname.lastname@epfl.ch

Abstract. The composition of services that are indexed in a large-scale service directory often involves many complex queries issued by the service composition algorithm to the directory. These queries may cause considerable processing effort within the directory, thus limiting scalability. In this position paper we present a novel approach to increase scalability: The directory offers a compact directory digest that service composition clients can download to solve the hard part of the composition problem locally. In order to obtain the final solution, only a reduced number of simple directory queries is needed.

Keywords: Service composition, service discovery, service directories

1 Introduction

There is a good body of work which addresses the service composition problem with planning techniques based either on theorem proving (e.g., Golog [6], SWORD [10]) or on hierarchical task planning (e.g., SHOP-2 [14]). All these approaches assume that the relevant service descriptions are initially loaded into the reasoning engine and that no discovery is performed during composition.

However, due to the large number of services and to the loose coupling between service providers and consumers, services are indexed in directories. Consequently, planning algorithms have been adapted to a situation where planning operators are not known a priori, but have to be retrieved through queries to these directories. In [11] the authors used a simple service composition algorithm based on forward chaining. However, service discovery and composition were not efficiently integrated, resulting in the discovery of large numbers of services. Lassila and Dixit [5] addressed the problem of interleaving discovery and composition, but they considered only simple workflows where services had one input and one output.

As current web service directories do not have semantically well-defined representations and thus are not well-adapted for automatic matchmaking, we focused on a directory for service descriptions based on OWL-S [9]. In [2] we presented a directory with a flexible interface that allows to specify matching conditions and ranking heuristics in order to optimize the interaction with different service composition algorithms. The service composition algorithm presented in [3] ensures the type-compatibility of formal and actual service parameters. In [1] the query processing algorithm used by the directory was refined to enable a best-first search, reducing the response time and processing effort within the directory.

However, despite of these advances, a single service composition may involve a large number of directory queries, and each query may have to process a significant part of the directory. E.g., in the case of service composition algorithms using forward chaining, a single directory query may require the processing of 20% of the directory data. I.e., all queries issued by a single service composition may together cause a workload in the directory that is proportional to the directory size. As the directory is a shared resource, it is evident that this kind of complex queries make the directory become a bottleneck. Massive replication of the directory service is needed for scalability reasons, which may be very expensive due to the large number of needed directory servers.

In this position paper we present a novel approach to service composition, which avoids complex directory queries. In our approach, the directory offers a compact digest that summarizes the input/output behaviour of available services. The service composition clients download the digest and use it to solve the hard part of the composition problem locally. Only simple directory queries are issued to obtain the final result.

This position paper is structured as follows: Section 2 introduces a simplified service description formalism and our definition of service composition. Section 3 gives an overview of service composition exploiting a directory digest. Section 4 sketches several ways to represent a directory digest. Section 5 outlines how service composition algorithms can be adapted to exploit a directory digest. Finally, Section 6 concludes this position paper.

2 Service Description and Service Composition

Service descriptions are a key element for service discovery and service composition, as they enable automated interactions between applications. In our system, a service S is described by two parameter sets – the *input parameters required by the service* S_{in} and the *output parameters provided by the service* S_{out}. Each parameter is identified by a unique name in its set. We assume that this name uniquely identifies the meaning (semantics) of the parameter. Despite its simplicity, our service description formalism is consistent with existing formalisms, such as WSDL [13] and OWL-S [9]. Part of the input/output specification of services described in WSDL or OWL-S can be mapped to our simplified formalism.

Following the terminology used in [4], *functional-level service composition* addresses the problem of selecting a set of services that – combined in a suitable way – are able to match given user requirements. Each existing service is defined in terms of an atomic interaction, i.e., in terms of its input and output parameters as well as of its preconditions and effects. In [12] the authors introduce *process-level service composition*, a promising approach that addresses complex service interactions and aims at generating executable code to implement a composed service. However, in this position paper we focus on functional-level service composition.

A service composition problem Q consists of a set of available input parameters Q_{in} and a set of required output parameters Q_{out}. The composed service, which is represented as a workflow, has to compute all required output parameters. Using forward chaining for service composition, a service S can be invoked only if all the service inputs are available ($S_{in} \subseteq Q_{in}$). Upon invocation, S generates the outputs S_{out}, which may be used as inputs for further service invocations. I.e., the application of S yields a new service composition problem Q', where $Q'_{in} = Q_{in} \cup S_{out}$ and $Q'_{out} = Q_{out}$. The service composition problem is solved if $Q'_{out} \subseteq Q'_{in}$. In previous work [1] we used complex directory queries in order to dynamically discover applicable services (i.e., services for which all required inputs are available) during service composition, which caused a very high workload within the directory and consequently also slowed down the service composition algorithm because of expensive remote interactions with the directory.

3 Service Composition with Directory Digest

The approach presented here increases scalability by avoiding complex directory queries during composition. The directory offers a compact digest that summarizes the input/output behaviour of the services indexed in the directory. Service composition clients download the digest, which contains sufficient information to compute a service composition locally. The interactions between the service composition client and the directory follows this scheme:

1. Download digest. The client periodically downloads the most recent version of the directory digest. As service descriptions usually remain constant for a longer period of time, the clients do not have to reload their copy of the digest for every service composition request. Typical refresh rates would be once per day, once per week, etc. Directory digests have version numbers, so the directory may support digest patches, i.e., incremental updates of the digest.
2. Transform composition problem Q. As the digest is a compressed representation of the input/output characterisations of indexed services, full parameter names are not available in the digest. Hence, the composition client sends Q to the directory and receives Q^T. The directory maps each parameter in Q_{in} resp. Q_{out} to the corresponding digest parameter in Q^T_{in} resp. Q^T_{out}. This translation is a simple mapping and can be processed in linear time with the number of parameters in Q.
3. Compute composition. Using the directory digest and Q^T, the client locally computes a service composition, without any queries to the directory. If the composition fails, the client may update its copy of the directory digest and retry (the new digest may cover the input/output behaviour of new, recently registered services).
4. Transform composition result. If the composition problem has been successfully solved in the step before, the client knows the exact input/output characterization of the selected services that are part of the composition workflow. It asks the service to provide service descriptions for the selected services. The resulting directory query looks only for *exact matches*, which can be processed very efficiently by the directory. E.g., the directory may simply use the input/output characterization to compute a hash key and look up the service descriptions in a hash table. If the desired input/output characterization is not found in the directory (i.e., services have been removed), the client has to download an up-to-date directory digest and retry.

4 Directory Digest Representation

The directory digest may be represented in many different ways. The simplest one is a bit matrix, where each column corresponds to a certain parameter and each row describes an available input/output characterization (i.e., for each row, the directory contains one or more services with the corresponding input/output behaviour). Assume there are n different parameter names used by service descriptions in the directory, which are identified by their index i ($0 \leq i < n$). The column position $2i$ corresponds to the i^{th} parameter used as input, while the position $2i+1$ corresponds to the i^{th} parameter used as output. This representation is extensible, i.e., a new parameter can be included by adding 2 columns.

As an example, assume we have the parameter names A (index 0), B (index 1), C (index 2), D (index 3) and 2 service descriptions S_1 and S_2 with the input/output behaviour $S_1 : \{B, C\} \to \{D\}$ resp. $S_2 : \{B\} \to \{C, D\}$. This can be represented by the following bit matrix:

Column index	0	1	2	3	4	5	6	7	Corresponding
Column meaning	A_{in}	A_{out}	B_{in}	B_{out}	C_{in}	C_{out}	D_{in}	D_{out}	service
	0	0	1	0	1	0	0	1	S_1
	0	0	1	0	0	1	0	1	S_2

For a large number of parameters and a large number of different input/output characterizations, the matrix may become quite large. For instance, assume we have 10^3 different parameter names and 10^6 different input/output characterizations. The resulting matrix has $2 * 10^9$ bits, i.e., about 238MB.

However, in practice, the matrix is sparse, i.e., each row has only a few bits set, because the number of service inputs/outputs is limited. Hence, a more compact representation can be obtained by listing the column indices of parameters that are present. If there are 1000 different parameter names, the column indices range from 0 to 1999, which can be represented by 11 bits. As a row delimiter, either an otherwise unused index (e.g., $2^{11} - 1$) may be defined or we add one extra bit to mark the end of a row. If we assume that on average each service has 3 inputs and 3 outputs and we use an extra bit to mark the end of rows, the matrix can be represented by $6 * 12 * 10^6$ bits, which is less than 9MB. Further reductions could be obtained e.g. by applying standard compression algorithms. In short, the size of the directory digest is not prohibitive.

If we consider each row in the matrix a combination and the whole matrix a combination set, we can also use a (serialized) 0-suppressed BDD (ZDD) as directory digest, which is known to be a highly efficient representation of a combination set [7].

5 Adaptation of Composition Algorithm

Now the question arises how the composition clients use the directory digest. If the directory digest is represented as a bit matrix, service composition can be implemented by simple bit operations. These operations can be implemented highly efficiently by exploiting e.g. special processor instructions, such as e.g. the MMX instructions of Intel processors.

$IMask$=(101010...) resp. $OMask$=(010101...) is the bit mask that selects all inputs resp. outputs of a row. Inp are the available inputs (corresponding to Q^T_{in}), $Outp$ are the required outputs (corresponding to Q^T_{out}), represented as bit vectors. In the following, & is the bitwise AND, $|$ the bitwise OR, and $>>$ the shift-right operation (inserting 0 on the left side). The composition is completed if (Inp & ($Outp >>$ 1)) = ($Outp >>$ 1). A service corresponding to a row R is applicable if (R & Inp) = (R & $IMask$). After application of the service, Inp is updated as follows: Inp := Inp $|$ ((R & $OMask$) $>>$ 1).

For the sparse matrix representation, a service composition algorithm using simple and efficient operations can be easily derived, too. If the directory digest is represented as a ZDD, we can leverage ZDD operations, in particular the 'permission operation' $\oslash$ [8], in order to select combinations that represent applicable services.

6 Conclusion

Service composition algorithms that dynamically retrieve possibly relevant service descriptions from a large-scale service directory tend to issue a large number of complex directory queries, causing high workload within the directory. As such an approach does not scale well, we introduced downloadable directory digests that allow service composition algorithms to solve the hard part of a composition problem locally without any interaction with the directory. Directory queries are only needed to translate the initial composition problem and to obtain the final result. These queries are very simple and require only a lookup in a table, significantly reducing the workload in the directory and boosting scalability.

References

1. W. Binder, I. Constantinescu, and B. Faltings. A directory for web service integration supporting custom query pruning and ranking. In *European Conference on Web Services (ECOWS 2004)*, pages 87–101, Erfurt, Germany, Sept. 2004.
2. I. Constantinescu, W. Binder, and B. Faltings. An extensible directory enabling efficient semantic web service integration. In *3rd International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, Nov. 2004.
3. I. Constantinescu, B. Faltings, and W. Binder. Large scale, type-compatible service composition. In *IEEE International Conference on Web Services (ICWS-2004)*, pages 506–513, San Diego, CA, USA, July 2004.
4. R. Lara, W. Binder, I. Constantinescu, D. Fensel, U. Keller, J. Pan, M. Pistore, A. Polleres, I. Toma, P. Traverso, and M. Zaremba. Knowledge Web research deliverable D2.4.2: Definition of semantics for web service discovery and composition. Web pages at http://knowledgeweb.semanticweb.org/, 2004.
5. O. Lassila and S. Dixit. Interleaving discovery and composition for simple workflows. In *Semantic Web Services, 2004 AAAI Spring Symposium Series*, 2004.
6. S. A. McIlraith and T. C. Son. Adapting Golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, Apr. 22–25 2002. Morgan Kaufmann Publishers.
7. S. Minato. Zero-suppressed BDDs for set manipulation in combinatorial problems. In A.-S. IEEE, editor, *Proceedings of the 30th ACM/IEEE Design Automation Conference*, pages 272–277, Dallas, TX, June 1993. ACM Press.
8. H. G. Okuno, S. ichi Minato, and H. Isozaki. On the properties of combination set operations. *Information Processing Letters*, 66(4):195–199, May 1998.
9. OWL-S. DAML Services, http://www.daml.org/services/owl-s/.
10. S. R. Ponnekanti and A. Fox. Sword: A developer toolkit for web service composition. In *11th World Wide Web Conference (Web Engineering Track)*, 2002.
11. S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In *Proceeding of the AAAI-2002 Workshop on Intelligent Service Integration*, pages 1–7, Edmonton, Alberta, Canada, July 2002.
12. P. Traverso and M. Pistore. Automated composition of semantic web services into executable processes. In *International Semantic Web Conference*, volume 3298 of *Lecture Notes in Computer Science*, pages 380–394. Springer, 2004.
13. W3C. Web services description language (WSDL) version 1.2, http://www.w3.org/tr/wsdl12.
14. D. Wu, B. Parsia, E. Sirin, J. Hendler, and D. Nau. Automating DAML-S web services composition using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, 2003.