Semantic Web Service Composition Framework Based on Parallel Processing*

Peter Bartalos and Mária Bieliková
Institute of Informatics and Software Engineering, Faculty of Informatics
and Information Technologies, Slovak University of Technology in Bratislava
{bartalos,bielik}@fiit.stuba.sk

Abstract

The process of semantic web service composition arranges several web services into one composite service to realize complex workflows with an exploitation of semantics. This paper proposes a framework to automatic semantic web service composition. Its advantage is that a huge amount of computation is performed during preprocessing and the composition approach is designed to exploit the parallel execution of processes which is nowadays supported in multiprocessor platforms. The framework is built to suit the WS Challenge competition requirements.

1 Introduction

Semantic web services present a topical research area aimed at exploiting semantic annotation of web service descriptions [14]. One of the most studied topics in this area is the automation of the semantic web service composition aiming at arranging several services into one complex service to be able to realize more complicated workflows. The composition is studied in different contexts to achieve practical applicability [1, 6, 9].

Several methods used in semantic web service composition based mostly on AI planning techniques were proposed. These include among others the state space search, graph based planning, Hierarchical Task Network (HTN) planning and approaches based on logic programming [11].

Even though a lot of work has been done in the field of semantic web service composition, there still remain problems. These are mainly related to insufficient automation of composition and execution, and scalability of approaches. Accordingly current research is more oriented to solve the issues related to scalability from performance point of view, QoS (Quality of Service) consideration and regarding the pre/postconditions of services.

Promising results dealing with scalability are presented in [7, 12]. The authors perform the composition in insignificant time (in msecs) even if the number of services rises to thousands. Both approaches realize preprocessing to decrease the computation required during the composition. They also consider pre/postconditions of web services. Other approaches considering pre/postconditions are described in [5, 13].

The research behind considering QoS attributes of web services during composition is motivated by the assumption that there are several web services having similar functionality. These can be alternatively used to accomplish the same goal. However, the services having similar functionality often differ in QoS attributes. Hence, to create the composition having the best aggregated QoS characteristic, the QoS of single services have to be considered. Some approaches dealing with QoS aware composition are presented in [4, 8].

In this paper we give an overview of our framework for semantic web service composition. It is designed to suit the requirements of the *WS Challenge* competition (`http://ws-challenge.georgetown.edu/wsc09/index.html`). We have adapted our approach described in [2] to fit the requirements of the competition. Our work is motivated also by the *SEMCO-WS* research project [10]. Our composition framework is here used to automate the process of a workflow creation in the context of a multi user workflow editor [3].

Our approach bases on the following issues:

- **Find all possible solutions.** We find all possible solutions and select the best based on the QoS.
- **Maximize preprocessing.** We perform a huge amount of computation in preprocessing phase and build a data structure allowing quick response to the user query.
- **Parallel processing.** We build our framework in such a way that it effectively exploits the possibilities of multiprocessor platforms.

*This work was partially supported by the Slovak Research and Development Agency under the contract No. APVV-0391-06, "Semantic Composition of Web and Grid Services" and the Scientific Grant Agency of Slovak Republic, grant No. VG1/0508/09.

2 Preparing Data for Composition

We represent the web service composition as a DAG (directed acyclic graph). The nodes represent web service invocations. There is an edge between two nodes if the web service invocation represented by the first node produces data which may by consumed during the invocation of the web service represented by the second node. In this case we say that the services can be chained. The edges are seen also as connectors between the outputs of the first service and the inputs of the second service. During the creation of the composition we depict whether the connection is considered to be alternative or parallel with other connections. Hence, one DAG can represent more than one (partially ordered) sequence of the web service invocation producing required data based on the given input data. Acquired solutions may differ in their QoS characteristics.

Our approach to web service composition is based on a preprocessing. During the preprocessing we create a data structure used to quickly answer two queries. The first query (Q_1) serves to select services providing data required in the user query. The second query (Q_2) selects services providing data required as input for a given service. These two queries are used to perform the web service composition. In the following, we present the used data structure and the process populating this data structure with data considering the given web services set.

Our data structure is based on a relational repository having two parts, see a scheme in Fig. 1. The first stores data about the I/O of services. After the first part is populated, we query it to evaluate which services can be chained. During the evaluation we consider also the subsumption relation between the I/O. If there is found a suiting I/O pair, we store this information in the second part.

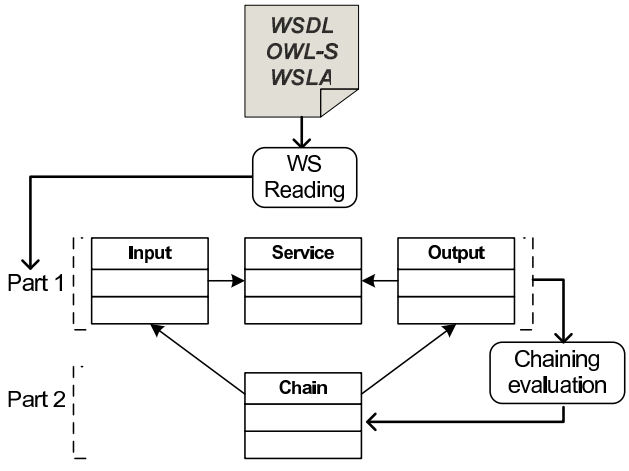

Figure 1. Data structure creation.

In the preprocessing phase we also analyze the given web service set and evaluate their characteristics. The result of the analysis consists in additional data exploited during the composition to speed it up. Our first analyzes results in a selection of those concepts for which there is no web service producing data annotated with an instance of that concept. If some service has at least one input annotated with an instance of such a concept, the only situation when this service can be used in the composition is if such input is provided in the user query. If it is not, we do not consider this service during the composition. If the unconsidered service is the only source providing input data for other services, we do not consider also these. This rule is applied recursively. We denote these services as *user data dependent services*.

After the user data dependent services are selected we rate them. We evaluate the effect of the case when the required data are not provided in the user query. The rating is higher, if an unusable service affects more services which cannot then be considered during the composition. Here we again apply this rule recursively on those services for which the only source of input data is the unusable service.

3 Composition Algorithm

The aim of our composition algorithm is to find effectively all possible solutions of the web services composition. It is based on the fact that only those services can be considered during the composition which have available input data. These are provided in the user query or as outputs of previously invoked services. Because of this, we mark the web services to denote if they can be used during the composition or not. At the beginning we do not know which services can be considered. Hence, we initially mark all services as *undecided*. If we realize that we have available all input data for some service, we mark it *usable*. On the other hand, if we realize that there is no source providing required input data for a service, we mark it *unusable*. Based on this, our composition approach is a combination of three separated processes operating over the same data structure: 1) Finding usable services, 2) Finding unusable services, 3) Backward chaining.

Finding usable services (see Algorithm 1) is based on forward chaining starting with those services for which we have input data from the user query. The aim of this process is to mark services as *usable* and build the composition from services having available input data.

The goal of the finding unusable services process is to restrict the set of services which are considered during the composition. It is based on the idea that if we remove unusable services, we improve the performance of the two other processes, because they will not waste time by processing such services. The process is executed over each service from the list of user data dependent services for which the inputs are not provided in the user query, (see Algorithm 2).

Here we consider also the rating of the services described in the Section 2. The algorithm runs with higher priority for services with higher rating, i.e. these are removed first. The idea is that if we first remove services with higher rating, we save more time because the other processes waste less time by processing unusable services and branches.

The aim of the backward chaining (see Algorithm 3) is to build the composition backward from the goal. First, services providing data required in the user query are selected by Q_1. Here the querying mechanism of relational databases is exploited to quickly select the required services. Then, Algorithm 3 is performed over each service with unsatisfied inputs. For each of them we look for an other web service producing them. This is performed by Q_2 over a runtime in-memory data structure.

Algorithm 1 FindUsableServices: Input: *service*

```
for all successor of service do
  if isUnusable(successor) == true then
    continue;
  end if
  set connectedInput of successor as provided;
  chain(service, successor);
  if areAllInputsProvided(successor) == true then
    mark successor as usable;
    FindUsableServices(successor);
  end if
end for
```

Algorithm 2 FindUnusableServices: Input: *service*

```
for all successor of service do
  if isOnlyInputSource(connectedInput, service) == true then
    mark successor as unusable;
    FindUnusableServices(successor);
  end if
end for
```

Algorithm 3 BackwardChaining: Input: *service*

```
for all input of service do
  if isProvided(input) == true then
    continue;
  end if
  S ← getProviders(input)
  for all provider in S do
    if isUnusable(provider) == true then
      continue;
    end if
    chain(provider, service);
    if isUsable(provider) == false then
      BackwardChaining(provider);
    end if
  end for
end for
```

The first two processes finish when there are no services having successors which have not been processed. The backward chaining ends when there is no service having input for which not each source providing the required data is checked. After finishing each process we traverse the chained services and create the solution. Here we consider the QoS attributes of services. We select the solution qualified as the best considering QoS.

4 Framework Architecture

The architecture of the composition framework is divided into two main subsystems, see Fig. 2. The first includes components responsible for the bootstrap phase. The second is responsible for the user querying phase. Each subsystem has access to a relational database storing information about the given web service set. The functionality of both subsystems is provided via a web service interface.

The *Bootstrap subsystem* is coordinated by the *System initializer*. Via the *WS interface* this component is directed to start the bootstrap process. During it, *System initializer* step by step calls the *WS Reader*, *Chaining evaluator* and *WS analyzer*. *WS Reader* is responsible for reading the service description files (wsdl, owl and wsla – QoS). It parses these files and stores the data in the relational database. The stored data are then processed to create data used during the composition. The *Chaining evaluator* decides which web service can be chained. The *WS analyzer* investigates the dependency of the services on the user input data and evaluates the related rating as discussed in the Section 2. The results of the chaining evaluation and web service analysis are stored in the respective parts of the relational database and in a runtime in-memory data structure. After the bootstrap finishes, the system is ready to answer the user queries.

The *User querying subsystem* is managed by the *Process manager*. Via the *WS interface* this central component provides the composition functionality. It is also responsible for calling a call back web service submitting the resulted composition. After receiving the user query, the *Process manager* manages *Composition realizer* to compute the composition. The *Composition realizer* is built in such a way that it runs different number of parallel threads to execute processes presented in the Section 3. The creation and running of the threads is controlled by the *Process manager*. It adapts the number of running threads based on the actual state of the composition process. The maximal number of threads is influenced by the hardware platform (number of processors/cores) in which the system is running. Each process operates over a single data structure managed by the *Data structure manager*. After the solution is found, *Solution generator* retrieves the solution from the data structure and serializes it into the required format (BPEL). After this, the call back web service is invoked to submit the solution.

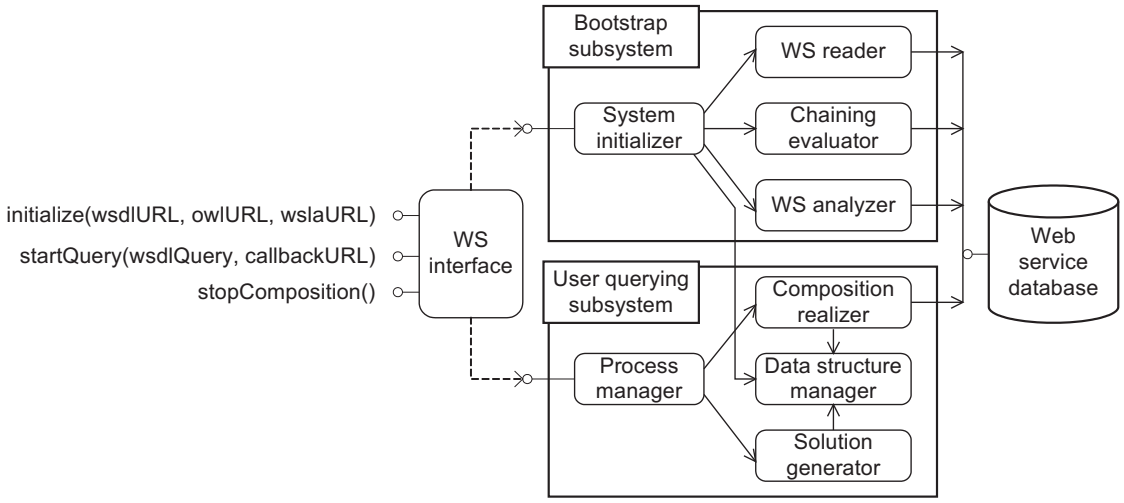

Figure 2. Composition framework architecture.

5 Conclusions and Future Work

In this paper we presented the concept of our framework for semantic web service composition. The composition algorithm is designed to effectively exploit the possibilities of multiprocessor platforms. The framework's architecture is adjusted to parallel processing over a single data structure.

Our future work deals with the problem of integrating service composition into applications from software engineering point of view. We study how the service composition can be effectively applied to automatic business process creation. This includes also an examination of the effects to the software architecture of the application.

References

[1] P. Bartalos and M. Bielikova. Enhancing semantic web services composition with user interaction. In *SCC '08: Proc. of the 2008 IEEE Int. Conf. on Services Computing*, pages 503–506. IEEE CS, 2008.

[2] P. Bartalos and M. Bielikova. Fast and scalable semantic web service composition approach considering complex pre/postconditions. In *WSCA '09: Proc. of the 2009 IEEE Congress on Services, Int. Workshop on Web Service Composition and Adaptation*. IEEE CS, 2009. Accepted.

[3] P. Bartalos, I. Kapustik, and V. Rozinajova. Visual support of workflow composition involving collaboration. In *GCCP '08: Proc. of the 2008 Int. Workshop on Grid Computing for Complex Problems*, pages 120–127. SAS, 2008.

[4] R. Berbner, M. Spahn, N. Repp, O. Heckmann, and R. Steinmetz. Heuristics for qos-aware web service composition. In *ICWS '06: Proceedings of the IEEE Int. Conf. on Web Services*, pages 72–82. IEEE CS, 2006.

[5] Y. Gamha, N. Bennacer, G. V. Naquet, B. Ayeb, and L. B. Romdhane. A framework for the semantic composition of web services handling user constraints. In *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*, pages 228–237. IEEE CS, 2008.

[6] L. Hluchy, O. Habala, M. Babik, M. Laclavik, Z. Balogh, and E. Gatial. Knowledge-based platform for environmental risk management. In *ISPDC '07: Proc. of the Sixth Int. Symposium on Parallel and Distributed Computing*, pages 3–9, Washington, DC, USA, 2007.

[7] S. Kona, A. Bansal, and G. Gupta. Automatic composition of semantic web services. In *ICWS '07: Proc. of the 2007 IEEE Int. Conf. on Web Services*, pages 150–158. IEEE CS, 2007.

[8] A. Liu, Q. Li, L. Huang, M. Xiao, and H. Liu. Qos-aware scheduling of web services. In *WAIM '08: Proceedings of the 2008 The Ninth Int. Conf. on Web-Age Information Management*, pages 171–178. IEEE CS, 2008.

[9] J. Paralic and M. Paralic. Some approaches to text mining and their potential for semantic web applications. *Information and Organizational Sciences*, 31(1):157–170, 2007.

[10] M. Paralic, O. Habala, J. Paralic, and P. Bartalos. Semantic composition of web and grid services. In *Znalosti 2009*, pages 355–358, Brno, 2009.

[11] J. Peer. *Web Service Composition as AI Planning – a Survey*. University of St.Gallen, 2005.

[12] K. Ren, X. Liu, J. Chen, N. Xiao, J. Song, and W. Zhang. A qsql-based efficient planning algorithm for fully-automated service composition in dynamic service environments. In *SCC '08: Proc. of the 2008 IEEE Int. Conf. on Services Computing*, pages 301–308. IEEE CS, 2008.

[13] A. Sirbu and J. Hoffmann. Towards scalable web service composition with partial matches. In *ICWS '08: Proc. of the 2008 IEEE Int. Conf. on Web Services*, pages 29–36. IEEE CS, 2008.

[14] R. Studer, S. Grimm, and A. Abecker. *Semantic Web Services: Concepts, Technologies, and Applications*. Springer, 2007.