Visualizing Compositions of Services from Large Repositories

Marco Aiello, Nico van Benthem, Elie el Khoury
Dep. of Mathematics and Computing Science
University of Groningen
Nijenborg 9 – 9747AG Groningen
The Netherlands
{aiellom@cs.,N.van.Benthem@student.,e.el.khoury@}rug.nl

Abstract

Creating a Service-Oriented Architecture requires the identification of services to be composed together in order to solve a given need. Currently, software engineers perform this design task by hand by identifying services and how to compose them. In this paper, we propose RuGCo a system to automatically compose Web services coming from a static large repository, according to a domain ontology expressed in OWL, on the basis of a user query. The proposed system is not only able to find all suitable combinations of Web services, but also offers a visualization of the composition to let the software engineer inspect all possibilities.

1. Introduction

As Service-Oriented Architectures are becoming more popular, software engineers are spending their time increasingly on developing services and making them interact. The process of making many different services cooperate while simultaneously following some user goal is usually referred to as *service composition*. Currently, developers base their work on the family of XML technologies known as Web services [6] and perform the composition by hand. That is, they consider the various services as building blocks of their software architecture and decide how to orchestrate them in order to have a working information system.

The more automation one can bring to the process, the more efficient would the production of software be. To this end, a number of research efforts go into the direction of automatically composing services based on some request from the user, may he be an expert engineer or an end-user (see, e.g., [5]). Naturally, in order to enable automatic composition a number of additional tools are necessary, most notably: (i) ontologies describing the relation among entities in the application domain, (ii) composition algorithms that transform the request into an execution, (iii) monitoring frameworks that control the execution.

Automating the composition does not necessarily mean removing the engineer from the picture. The engineer should be involved in the process and interact with the system in order to make design choices while ignoring low level details such as checking for semantic compatibility of operations or availability of services. A visual interaction with such an automatic service composition system would finally be the ideal approach to empower the engineer while keeping the control task pleasant and effective.

In this paper, we present RuGCo, a system to automatically compose Web services coming from a large repository based on a domain ontology and a user query. The system also offers a visualization of the compositions found to the user. RuGCo is the entry from the University of Groningen to the 2008 Web service challenge.

The remainder of the paper is organized as follows. Section 2 contains an overview of the Web-service challenge for which an entry is presented in the following Section 3. Section 4 contains insights on how the system is evaluated. Section 5 offers concluding remarks on the presented work.

2. Challenge Definition

The Web-Service Challenge[1] is a yearly venue for researchers to compare Web service composition solutions [3]. The emphasis of the challenge is that of working with large static repositories of Web services and with a semantic taxonomy of service operation names. Entries to the challenge, which is in its fourth edition, are judged based on the completeness of the solution, the quality of the architecture and the speed of finding all compositions.

The competing systems use the following data as input:

1. A Web Service Description Language (WSDL) file containing a set of services interface descriptions along

[1]`http://cec2008.cs.georgetown.edu/wsc08/`

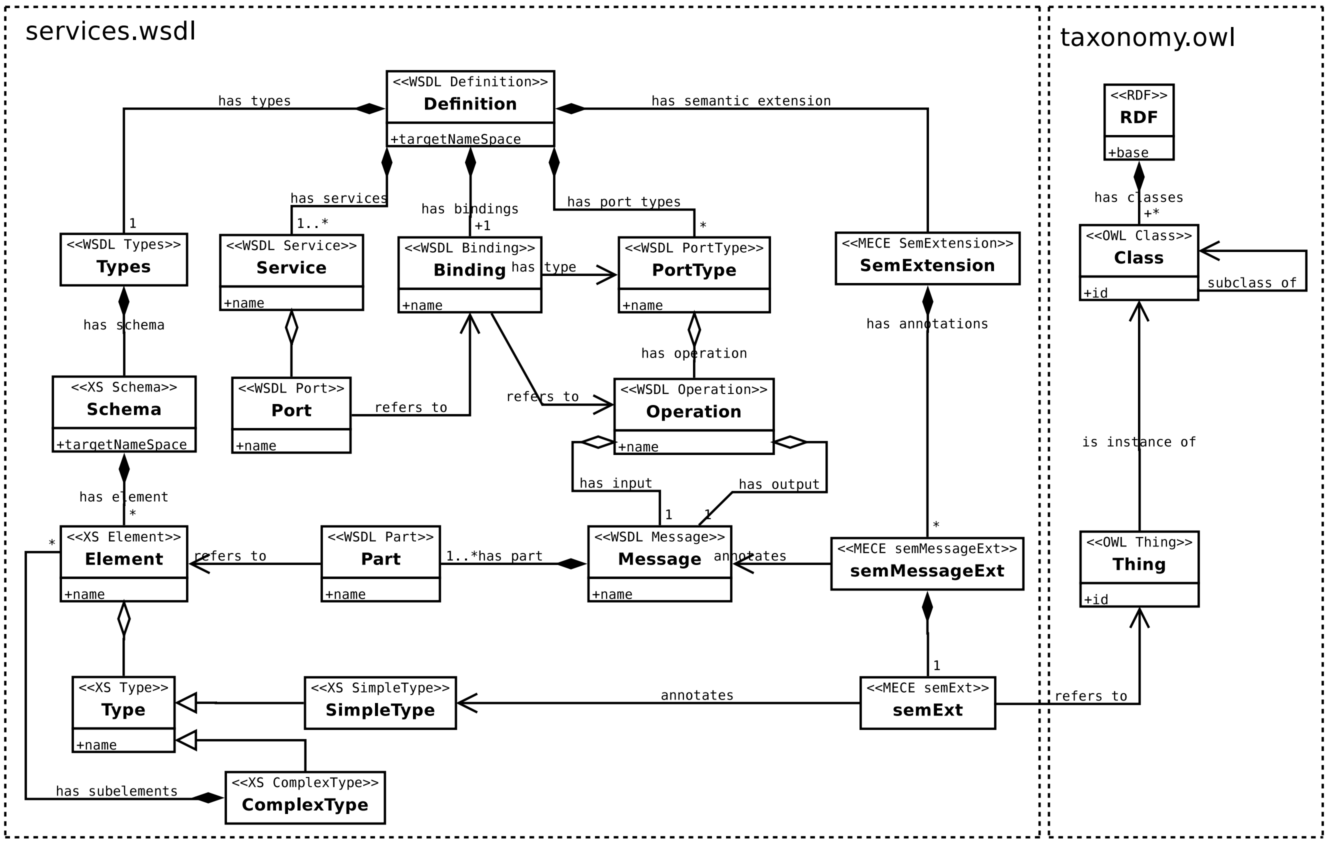

Figure 1. Metamodel of the service repository and ontology

with semantic annotation of their message structures. The number of services can vary greatly.

2. A Web Ontology Language (OWL) file containing a taxonomy of concepts relating the classes to be used in dealing with the service descriptions.

3. One or more queries to the system to be run against the repository and the ontology. The queries are a set of input names and output names that, in general, require a composition of services to be obtained.

The output required from the system is a Business Process Execution Language for Web Services (BPEL) schema of all compositions solutions, that is, a description of every possible execution path of services that starting from the given query input will finally provide the query output. The speed of finding all these compositions is judged without considering the time necessary to process the service repository description and ontology.

To gain a better understanding of the entities involved in the challenge, let us consider a metamodel of the services repository and its relation to the ontology, see figure Figure 1. Each `Service` interface defines a single operation linked to an and an output `Message`. Only simple elements within a message structure are annotated linking the element to a semantic individual (`Thing` entity), which is an instance of an ontology `Class`. Complex elements must be reasoned by it subelements. The ontology is RDF based and defines an hierarchy of classes.

3. The RuGCo system

The Rijksuniversiteit Groningen Web service Composition entry, RuGCo for short, is built of the following subsystems: (i) XML parsers needed for the service repository (a WSDL file) and for the ontology information (the OWL specification). The parsers are also responsible for generating the indexes from the parsed data; (ii) a composition engine based on a backward depth first search algorithm optimized for the challenge; (iii) a BPEL code generator used to convert the compositions found by the engine into one generic business process ready for execution.

Let us now consider the data processing flow of RuGCo, referring to Figure 2 from left to right. At the *Data* level, one has three sets of data (ovals in the figure): the services descriptions in WSDL format, the taxonomy in OWL for-

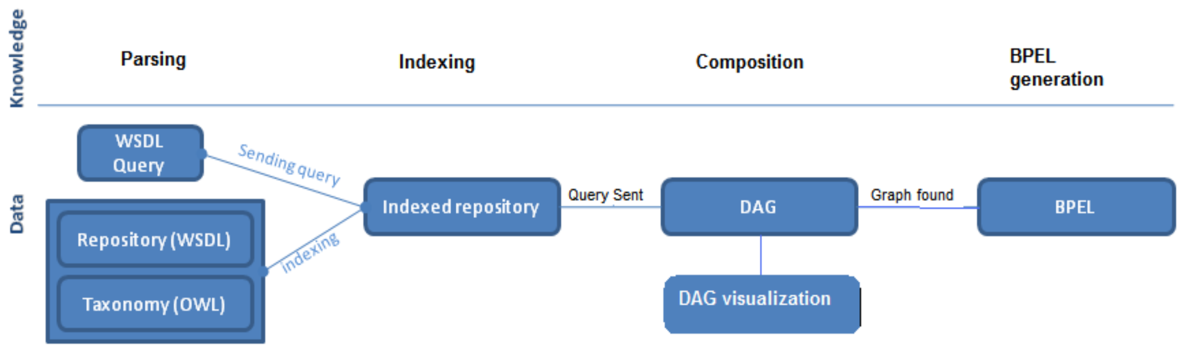

Figure 2. Data processing flow of RuGCo

mat, and the query also in WSDL format. At the *Knowledge* level, this translates in information to be parsed. The result of parsing is then an index data structure based on the input data. Using the query, the indexes are used to find the composition. The composition is represented by a (multi-) directed acyclic graph (DAG) where nodes are services and edges are input-output connections over service parameters. The connections are considered such in accordance with the given ontology. Such a directed graph is used for two purposes: (i) to show all the available compositions to the user via a graphical rendering; (ii) to output a complete BPEL specification as required by the challenge specification.

We have presented parsing and index data structures in [1], here we concentrate on the composition algorithm and the visualization component. Another approach using DAGs is presented in [4].

3.1 The composition algorithm

Given an index over a static set of Web service interfaces, composition becomes a matter of search. Given the challenge requirements, the issue is to find the most efficient search strategy in terms of execution time and space usage. We resort to well-known search strategies in the field of artificial intelligence [7], in particular we consider a variant of the Depth First Search (DFS) [2] Algorithm with an additional heuristic similar to [8]. The heuristic takes into account the likelihood that a branch will lead to a valid composition or not, measured in terms of the number of alternative candidate services for an input.

The algorithm builds a directed acyclic multi-graph in a backward fashion. It begins by the output query parameters and it then considers a service having one or more of the parameters as output. If found, the search continues with the inputs of the found service, otherwise the branch is discarded and the search continues on a new candidate service. If the inputs to the service are the ones given in the query, the search for that branch is considered completed and a new branch is considered. Next the pseudo-code.

```
findCompositionFor (message $M$)
Data   : Repository $R$; Query input message $M_q$; unfit services $U$;
         visited services $V$;
Output : Composition $C$
$C \leftarrow$ empty graph;
$s \leftarrow$ service of message $M$;
preorder elements of message $M$ using heuristic;
foreach element $e$ of $M$ do
    if $e$ not provided by $M_q$ then
        $foundProvider \leftarrow$ false ;
        foreach provider $p$ of $e$ in $R \setminus U$ do
            if $p$ not in $V$ or then
                add $p$ to $V$;
                $M_p \leftarrow$ input message of $p$);
                $C_p \leftarrow$ findCompositionFor ($M_p$);
                if $C_p \neq null$ then
                    $foundProvider \leftarrow true$;
                    add $C_p$ to $C$;
                end
            else
                $foundProvider \leftarrow true$;
            end
        end
        if not $foundProvider$ then
            add $s$ to $U$;
            return $null$
        end
    end
end
add s to $C$;
return $C$;
```

3.2 Visualizing compositions

After execution of the composition algorithm, a directed acyclic multi-graph representing all compositions that satisfy the query over the given service repository in accordance to the given ontology is produced. This is the output of RuGCo that then is translated into the BPEL format.

Since it is the main contribution of the system, it is also important to give the user the possibility to inspect it and consider the possible alternative compositions that have been found. To this end, a visualization of the data structure is of great advantage to the human operator. The visualization is much more useful as it allows to compactly but representatively show the data structure at hand. Furthermore, it should allow to see the dynamics of the system meaning to see how the system is executed and the consequence of

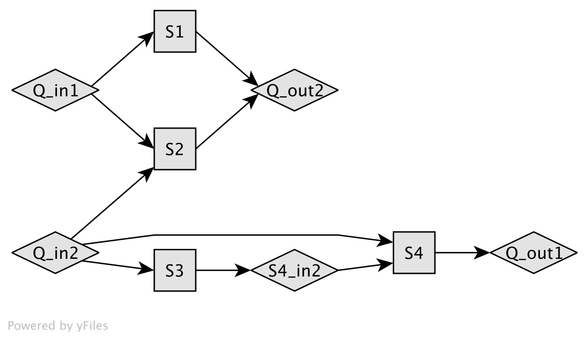

Figure 3. Visualization of a DAG

the user choices. We have not implemented such features yet, but in this initial phase have built a proof of concept based on an off-the-shelf component for rendering graphs : the *yFiles* — Java Graph Layout and Visualization Library[2] and on the graphML language[3]. An example of visualizing a composition is shown in Figure 3, where the query input is at the left and the final outputs at the right. Nodes are labeled by the service name and edges are labeled and grouped by the input elements of the service.

It should be noted that current BPEL engines also have visualization facilities, but our aim here is different. Namely, we want to visualize all available compositions before translating them into any specific format and show all the alternative paths.

4. Evaluation

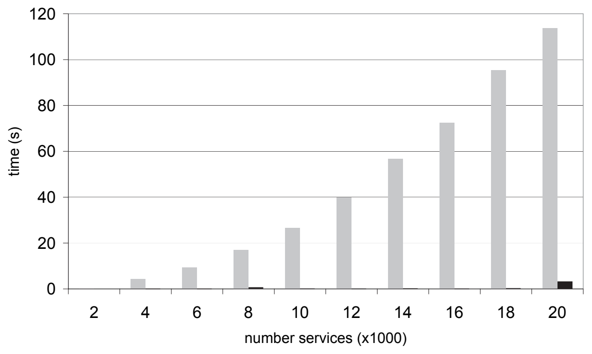

Figure 4. Evaluation of RuGCo

We tested RuGCo on a set of randomly generated services and ontologies to get an impression of the performance. The setup for the evaluation was the following: a standard PC equipped with AMD 3800 X2 dual core, 2 Gb 800 Mhz dual channel RAM, OS 64 bit Ubuntu 8.04. The implementation is in Java version 1.6 with initial and maximal heap size set to 1Gb.

In Figure 4, we report the experiment of running RuGCo on a set of composition of increasing size in the number of services (horizontal axis) keeping fixed the number of concepts to 100.000 and the depth of the searched solution to 30 services. On the vertical axis we report the average time in seconds over five runs to find the solution (gray bar) or to establish that there is no solution (black bar).

5. Concluding Remarks

We presented RuGCo, the University of Groningen 2008 entry to the Web service challenge. RuGCo is a system that performs service composition using a backward depth first search and visualizes all found compositions. The current version of RuGCo complies with the challenge specification, and we consider visualization as a key element in our work, since it will show all possible solutions in a graph. However we foresee a number of improvements and future work which include: (i) tuning of the composition algorithm to increase performance; (ii) using more sophisticated visualization techniques; (iii) enabling user interaction to choose a composition and guiding execution; (iv) enabling interaction between the user and the graphs, in a way to customize the resulting composition according to the user's needs.

References

[1] M. Aiello, C. Platzer, F. Rosenberg, H. Tran, M. Vasko, and S. Dustdar. Web service indexing for efficient retrieval and composition. In *CEC-EEE 2006*, pages 424–426. IEEE, 2006.

[2] R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[3] M. Blake, K. Tsui, and A. Wombacher. The eee-05 challenge: A new web service discovery and composition competition. In *EEE 2005*, pages 780–781. IEEE Computer Society, 2005.

[4] S. Kona, A. Bansal, G. Gupta, and T. Hite. Semantics-based web service composition engine. In *CEC-EEE 2007*. IEEE, 2007.

[5] A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. *Journal on Digital Libraries*, 6(3):235–246, 2006.

[6] M. Papazoglou. *Web Services: Principles and Technology*. Pearson, 2007.

[7] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach (2nd Ed.)*. Prentice Hall, 2002.

[8] Y. Zhang, M. Panahi, K. Raman, and K. Lin. Heuristic-based service composition for business processes with branching and merging. In *CEC-EEE 2007*, pages 525–528. IEEE, 2007.

[2]`http://www.yworks.com/`
[3]`http://graphml.graphdrawing.org/`