Web Semantics: Science, Services and Agents on the World Wide Web 17 (2012) 12–24

Contents lists available at SciVerse ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Improving semantic web services discovery using SPARQL-based repository filtering

José María García *, David Ruiz, Antonio Ruiz-Cortés

University of Seville, ETSI Informática, Av. Reina Mercedes, s/n, 41012 Sevilla, Spain

ARTICLE INFO

Article history:
Received 28 August 2010
Received in revised form
19 May 2012
Accepted 9 July 2012
Available online 24 July 2012

Keywords:
Semantic web services
Service discovery
Scalability
Service repositories
Semantic web query languages

ABSTRACT

Semantic Web Services discovery is commonly a heavyweight task, which has scalability issues when the number of services or the ontology complexity increase, because most approaches are based on Description Logic reasoning. As a higher number of services becomes available, there is a need for solutions that improve discovery performance. Our proposal tackles this scalability problem by adding a preprocessing stage based on two SPARQL queries that filter service repositories, discarding service descriptions that do not refer to any functionality or non-functional aspect requested by the user before the actual discovery takes place. This approach fairly reduces the search space for discovery mechanisms, consequently improving the overall performance of this task. Furthermore, this particular solution does not provide yet another discovery mechanism, but it is easily applicable to any of the existing ones, as our prototype evaluation shows. Moreover, proposed queries are automatically generated from service requests, transparently to the user. In order to validate our proposal, this article showcases an application to the OWL-S ontology, in addition to a comprehensive performance analysis that we carried out in order to test and compare the results obtained from proposed filters and current discovery approaches, discussing the benefits of our proposal.

1. Introduction

Current Semantic Web Services (SWS) discovery solutions often suffer from scalability issues, so large and complex service repositories cannot be properly handled by them. Although the research community is putting efforts into improving discovery mechanisms, the underlying reasoning facilities do not scale well in general [1]. The approach taken in this paper does not consist of yet another discovery mechanism, but on the inclusion of a preprocessing stage that filters service repositories using two different queries, so that the search space for discovery processes is reduced in our experiments, on average, from 12.5% of the original repository size up to 1.1%, depending on the concrete query used and the nature of the repository and user request. Consequently, service discovery execution time is greatly improved, performing the whole process, when using our proposed filters, at least 9.1 times faster and up to 44.7 times faster, with a contained penalty on precision, depending on each corresponding query and the underlying discovery mechanism chosen.

The number of currently available services in public repositories[1] is expected to explode in the future, so that billions of services will be able to be consumed on the Web [2]. Furthermore, currently available semantic descriptions, in terms of SWS classical ontologies such as OWL-S or WSMO, present a high complexity for defining and processing them. Both issues lead to a scenario where discovery mechanisms based on different logic formalisms have scalability issues. Consequently, current research efforts focus on providing improvements and optimizations of those mechanisms, using lightweight semantic technologies, in order to enhance the usability of SWS [3,4].

In order to alleviate the scalability problem on semantic discovery mechanisms, there are some proposals that provide different techniques to improve the discovery performance, such as indexing or caching descriptions [5], using several matchmaking stages [6], and hybrid approaches that include non-semantic techniques [7]. Our proposal takes a novel approach of reducing the input for discovery mechanisms, so that the resulting process is more streamlined, only reasoning about services which actually matter with respect to the user request. Thus, our solution filters services that can be discarded a priori, because they are not related at all

* Corresponding author. Tel.: +34 9545 59814; fax: +34 9545 57139.
E-mail addresses: josemgarcia@us.es (J.M. García), druiz@us.es (D. Ruiz), aruiz@us.es (A. Ruiz-Cortés).
URL: http://www.isa.us.es/josemaria.garcia (J.M. García).

[1] At the moment of writing, *seekda!* service crawler has indexed 28,606 services, *ProgrammableWeb* has registered 3,287 web APIs, and *iServe* repository contains 2,193 SWS descriptions.

1570-8268/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.websem.2012.07.002

with requirements and preferences stated by the user, considerably reducing the search space before actual discovery.

For example, consider the following scenario: a semantic service repository contains thousands of services from several travel-related domains, such as hotel bookings, plane tickets, car rentals, and travel insurances. If a user looks for a service that returns hotels given a particular city and a country, it is not necessary to process the whole repository to discover candidate services for the user request, but only consider the portion of services that are specifically related to the hotel lookup domain concepts that appear on the request, in this case. Thus, using lightweight technologies to preprocess the repository, the search space can be reduced in order to save computational resources and improve discovery performance.

For the proposed preprocessing, our proposal analyzes the user request in order to extract the concepts that are being used in its semantic definition (in the above example, some of them could be *City*, *Country* or *Hotel*, for instance). Then, the repository is filtered so that only services that use those concepts or related ones are selected to become the input for the subsequent discovery process (e.g. services whose definitions refer to *City*, *Country* and/or *Hotel* concepts, in the latter case).

Two different SPARQL [8] queries perform the filtering in our approach, namely $\mathcal{Q}_{all}$ and $\mathcal{Q}_{some}$. The former returns only those services whose definitions contain *all* the concepts referred by a user request, assuming that services have to fulfill every term of the request in order to be useful for the user. In turn, the latter query selects service definitions that refer to *some* (at least one) of the concepts referred by a user request, assuming that those services may satisfy its requirements and/or preferences to some extent, despite the missing information.

Our solution does not pretend to provide yet another discovery mechanism, but to introduce a preprocessing filtering stage, based on an accepted standard, that yields a notable improvement on heavyweight semantic processes, such as matchmaking of services. Furthermore, our proposed filtering does not add a noticeable amount of execution time with respect to matchmaking, because SPARQL queries used present a linear complexity on the size of the dataset and graph patterns included [9].

To the best of our knowledge, there are no proposals on filtering semantically-enhanced service repositories, but it is acknowledged that some sort of preprocessing can alleviate discovery and ranking tasks performed on those repositories [6]. To sum up, the main contributions of the proposal presented in this article are the following:

1. We propose a technique to improve semantic service discovery performance, based on a preprocessing stage that filters repositories in order to reduce the search space of subsequent discovery processes.
2. Our proposal is applicable to any discovery mechanism because it is performed before actual discovery occurs, and it allows interoperability with existing service repositories. In this work, we use the OWLS-MX hybrid matchmaker [7] to illustrate this point, though our proposal has also been applied to other discovery mechanisms [10].
3. Filtering is performed automatically from user requests, analyzing them and obtaining standard SPARQL queries without user interaction. Two different queries are presented, enabling two filtering levels, depending on the user needs and the characteristics of service repositories. We analyze and thoroughly discuss each query throughout the article.
4. In order to assess the actual impact of our proposal, we carried out a comprehensive, experimental study. Using a widely-used test collection (OWLS-TC), we applied our proposed filters to several discovery mechanisms, evaluating and discussing performance improvements using the Semantic Web Service Matchmaker Evaluation Environment (SME^2).

The rest of the article is structured as follows. Firstly, Section 2 presents some background information to contextualize and motivate the proposal. In Section 3 we show how to use SPARQL-based filtering within a discovery scenario, presenting both restrictive and relaxed filters that can be applied in different cases. Section 4 discuss the integration and implementation of our proposal applied to SWS frameworks, specifically OWL-S. Then, in Section 5 the performed experimental study is explained, analyzing the results and discussing the advantages of our proposal. Section 6 outlines the related work on this field. Finally, in Section 7 we discuss the conclusions.

2. Background

Using a Semantic Web query language is a natural fit for performing SWS discovery and ranking processes in terms of user requests, because, essentially, these processes search for elements in some sort of persistent storage using selection and ordering criteria. However, current query languages present shortcomings with respect to the level of inference and computation needed for SWS discovery and ranking. In the following we introduce the background elements of our proposal in order to contextualize and further motivate our work.

2.1. Querying the Semantic Web

There are three main approaches for Semantic Web query languages: graph-based, rule-based, and DL-based query languages [11–13]. Firstly, graph-based query languages allow us to fetch RDF [14] triples based on matching triple patterns with RDF graphs. Secondly, rule-based query languages propose logic rules to define queries, supporting RDF reasoning systems. Finally, DL-based query languages allow us to query Description Logic (DL) ontologies described in OWL-DL [15], being able to search for concepts, properties, and individuals. In general, rule- and DL-based query languages provide more reasoning mechanisms than graph-based ones, though it depends on the entailment regime applied to the concrete triple store and querying system. However, the former are not mature enough and they are in early stages of development [11], so the latter are more widely used, especially SPARQL [8], which is the current W3C Recommendation.

There are several graph-based query languages with different features [11], but SPARQL is the only language that is a W3C Recommendation [8]. In fact, it is fully supported in several implementations.[2] As a consequence, SPARQL (and its extensions) is the most widely used query language for the Semantic Web. There are several SPARQL implementations, such as Virtuoso, Sesame and ARQ,[3] which is included in the Jena Semantic Web Framework for Java. The latter is the chosen one for our evaluation tests presented in Section 5.

SPARQL, as a graph-based query language, explicitly accounts for the definition of labeled directed graphs by RDF triples, which conforms the very foundations of a Semantic Web ontology. Its main approach to query semantic repositories is to define graph patterns involving triple patterns, matching RDF triples, which are usually denoted (s, p, o), where s is the subject, p the predicate, and o the object. In order to work with said repositories, SPARQL has four different types of queries: `SELECT`, `CONSTRUCT`, `DESCRIBE` and `ASK`. Each type serves for a different purpose: `SELECT` queries return variables and their bindings with respect to the stored RDF triples; `CONSTRUCT` queries build an RDF graph based on

[2] http://www.w3.org/2001/sw/DataAccess/tests/implementations.

[3] Virtuoso: http://www.openlinksw.com/virtuoso/ ; Sesame: http://www.openrdf.org/ ; ARQ: http://jena.sourceforge.net/ARQ/.