Caching for Semantic Web Service Discovery

Michael Stollberg

Digital Enterprise Research Institute (DERI),
University of Innsbruck, Austria
michael.stollberg@deri.org

Abstract. This document is an extended abstract on a PhD work that develops an efficient, scalable, and stable Web service discovery engine. These qualities become important for discovery engines that serve as a software component in automated SOA technologies. Based on a profound formal specification, the approach is to capture design time discovery results and then use this knowledge for efficient runtime discovery. The work is evaluated by a statistical time efficiency comparison with other Web service discovery engines, and by a applicability study in real-world SOA applications.

Keywords: Semantic Web Services, Goals, Functional Descriptions, Discovery, Efficiency, Scalability, Stability

1 Introduction

Discovery is one of the central reasoning tasks in SOA systems, concerned with the detection of usable Web services for a specific request or application context. Aiming at the automation of this task, most existing works on semantically enabled Web service discovery focus on the quality of the applied matchmaking techniques. However, the following qualities become important for using an automated Web service discovery engine as a reliable software component in a SOA system: *efficiency* as the time required for finding a usable Web service, *scalability* as the ability to deal with large numbers of available Web services, and *stability* as the behavioral constancy among several invocations.

My PhD work addresses this challenge by applying the concept of caching to Web service discovery. For this, I extend the goal-driven approach that is promoted by the WSMO framework (`www.wsmo.org`). I distinguish *goal templates* as generic objective descriptions and *goal instances* as instantiations of a goal template that denotes concrete client requests. At design, Web service discovery for goal templates is performed. The result is stored in a graph that organizes goal templates by their semantic similarity and captures the minimal knowledge on the usable Web services for each goal template. This knowledge is utilized for efficient runtime discovery, i.e. the detection of a usable Web service for solving a goal instance that is defined by a client. In particular, this is achieved by:

1. *pre-filtering* as only the Web services that are usable for the corresponding goal template are potential candidates for the goal instance, and
2. *minimal use of a reasoner* for matchmaking because in certain situations the usability of a Web service for a goal instance can be directly inferred.

2 Solution Overview

My work extends the approach for Web service discovery promoted by the WSMO framework with a refined goal model and a rigid formalization for the functional aspects of Web service discovery. On this basis, the so-called *Semantic Discovery Caching* technique (short: SDC) caches the minimal knowledge in order to optimize the computational qualities of Web service discovery.

2.1 Web Service Discovery Framework

Figure 1 shows the conceptual model as a dataflow diagram. It deals with three entities: *Web services* that have a formal description and are accessible via a WSDL interface, *goal templates* as formalized, generic objective descriptions that are stored in the system, and *goal instances* that formally describe a concrete request by instantiating a goal template with concrete inputs. At design time, Web services for goal templates are discovered. The result is cached in the SDC graph, the knowledge structure for optimizing the Web service discovery process. At runtime, a concrete client request is formulated as a goal instance. The runtime discovery finds one usable Web service for solving this. It uses the cached knowledge for optimization, in particular for pre-filtering and minimizing the number of necessary matchmaking operations.

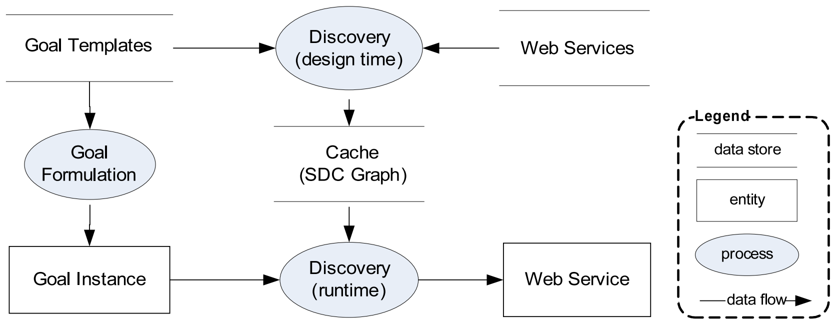


Fig. 1. Overview of Web Service Discovery Framework

In contrast to an invocation request for a Web service, a goal formally describes a client objective of getting from the current state of the world into a state wherein the objective is satisfied. This provides an abstraction layer for facilitating problem-oriented Web service usage: the client merely specifies the objective to be achieved as a goal, and the system discovers, composes, and executes suitable Web services for solving this. The distinction of goal templates and goal instances allows to better support the goal formulation by clients (e.g. by form-based instantiation through a graphical user interface), and – more importantly – provides the foundation for the two-phase Web service discovery outlined above.

I consider functional aspects as the primary aspect for discovery: if a Web service does not provide the functionality for solving a goal, then it is not usable and other, non-functional aspects are irrelevant. For this, the possible solution for goals and possible executions of Web services are formally described by functional descriptions $\mathcal{D} = (\Sigma, \Omega, IN, \phi^{pre}, \phi^{\mathit{eff}})$; Σ is the signature, Ω are domain ontologies, IN are the input variables, the precondition ϕ^{pre} and the effect ϕ^{eff} constraint the start- and end states. As the design time discovery result, the usability of a Web service W for a goal template $\mathcal{G}$ is expressed in terms of matching degrees (*exact, plugin, subsume, intersect, disjoint*). A goal instance is defined as a pair $GI(\mathcal{G}) = (\mathcal{G}, \beta)$ with the corresponding goal template $\mathcal{G}$ and an input binding β that is used to invoke a Web service W for solving $GI(\mathcal{G})$. If W is usable for $\mathcal{G}$ under the degrees *exact* or *plugin*, then W is also usable for any $GI(\mathcal{G})$; under the degrees *subsume* and *intersect*, additional matchmaking is required at runtime; if W is not usable for $\mathcal{G}$ it is also not usable for $GI(\mathcal{G})$.

2.2 Semantic Discovery Caching

The main contribution of my work is the SDC technique as the solution for enabling efficient, scalable, and stable Web service discovery. Its purpose is to improve the computational quality of the runtime discovery process by exploiting the relationships between goal templates, goal instances, and Web services.

The central element is the SDC Graph that provides an index structure for efficient search of goal templates and usable Web services. It organizes goal templates with respect to their semantic similarity, and keeps the minimal knowledge on the usability of the available Web services. Two goal templates $\mathcal{G}_i$ and $\mathcal{G}_j$ are considered to be similar if they have at least one common solution; if this is given, then mostly the same Web services are usable for them. In consequence, the upper layer of a SDC graph is the *goal graph* that organizes goal templates in a subsumption hierarchy, and the lower layer is the *usability cache* that captures the minimal knowledge on the usability of the available Web services. Upon this cache structure, the discovery operations make use of inference rules between the similarity degree of goal templates and the usability degree of Web services.

For illustration, Figure 2 shows an example of an SDC graph along with the most relevant inference rules. This considers three goal templates: $\mathcal{G}_1$ for package shipment within Europe, $\mathcal{G}_2$ for Switzerland, and $\mathcal{G}_3$ for Germany. As each solution for $\mathcal{G}_2$ is also a solution of $\mathcal{G}_1$, their similarity degree is *subsume*; the same holds between $\mathcal{G}_3$ and $\mathcal{G}_1$. These relationships are expressed by directed arcs in goal graph. Besides the goal templates, let there be some Web services, among them e.g. W_1 that provides package shipment within Europe, W_2 throughout the whole world, W_3 within the European Union, and W_4 within the Commonwealth. Their usability degree for each goal template is explicated by directed arcs in the usability cache. This knowledge is efficiently used for runtime discovery. Consider a goal instance for shipping a package from Munich to Berlin: its corresponding goal instance is $\mathcal{G}_3$; because W_1, W_2, and W_3 are usable for $\mathcal{G}_3$ under the *plugin* degree, we know that each of them is usable for solving the goal instance without the need of a matchmaker during runtime discovery.

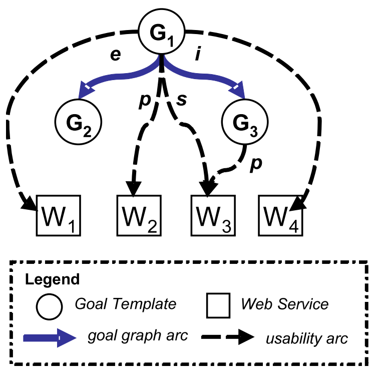
Structure of an SDC Graph	inference rules for $subsume(\mathcal{G}_1, \mathcal{G}_2)$
 G_1 e i p s G_2 G_3 p W_1 W_2 W_3 W_4 Legend Goal Template Web Service goal graph arc usability arc	(1) $exact(\mathcal{G}_1, W) \Rightarrow plugin(\mathcal{G}_2, W)$. (2) $plugin(\mathcal{G}_1, W) \Rightarrow plugin(\mathcal{G}_2, W)$. (3) $subsume(\mathcal{G}_1, W) \Rightarrow exact(\mathcal{G}_2, W)$ or (4) $subsume(\mathcal{G}_1, W) \Rightarrow plugin(\mathcal{G}_2, W)$ or (5) $subsume(\mathcal{G}_1, W) \Rightarrow subsume(\mathcal{G}_2, W)$ or (6) $subsume(\mathcal{G}_1, W) \Rightarrow intersect(\mathcal{G}_2, W)$ or (7) $subsume(\mathcal{G}_1, W) \Rightarrow disjoint(\mathcal{G}_2, W)$. (8) $intersect(\mathcal{G}_1, W) \Rightarrow plugin(\mathcal{G}_2, W)$ or (9) $intersect(\mathcal{G}_1, W) \Rightarrow intersect(\mathcal{G}_2, W)$ or (10) $intersect(\mathcal{G}_1, W) \Rightarrow disjoint(\mathcal{G}_2, W)$. (11) $disjoint(\mathcal{G}_1, W) \Rightarrow disjoint(\mathcal{G}_2, W)$.

Fig. 2. Example of a SDC Graph and Inference Rules

The SDC graph during its life time are maintained by algorithms that handle the addition, removal, and modification of goal templates and Web services. Two refinements ensure that the SDC graph exposes sophisticated search properties: (1) the only similarity degree that occurs in the goal graph is *subsume*, and (2) the minimization of the usability cache in order to avoid redundancy. The SDC technique is implemented as a discovery component of the WSMX system, available at the SDC homepage: `members.deri.at/~michaels/software/sdc/`.

3 Evaluation

To demonstrate the achievable quality increase for Web service discovery, I have run several comparison test between the SDC-enabled runtime discovery and an engine that applies the same matchmaking techniques but does not make use of the cached knowledge. Table 1 shows a snapshot of the statistical prepared test results; details and the original test data are available from SDC homepage. This clearly shows that the SDC discovery is **efficient** (the average time is always lower), **scalable** (the time for the SDC discovery remains the same for increasing numbers of Web services), and **stable** (the standard deviation is significantly smaller than the one of the comparison engine).

Another relevant aspect is the appropriateness of the assumptions that underly the conceptual model. For this, I have examined the applicability in real-world settings – e.g. in one of the world's largest SOA systems at telecommunication provider *Verizon*. In summary, there are many Web services that provide similar functionalities but differ in the detailed usage conditions. Also, the usage requests posted by the consuming applications can be expressed in terms of goals; these can be organized in a fine-grained subsumption hierarchy in the SDC graph so that its benefits for efficient runtime discovery can be exploited. Besides, the distinction of goal templates and goal instances has been regarded by practioneers as suitable way for realizing problem-oriented Web service usage.

Table 1. Comparison Test Statistics (all values in seconds)

No. of WS	engine	mean μ	median $\bar{x}$	standard deviation σ
10	SDC	0.28	0.27	0.03
	non-SDC	0.41	0.39	0.21
100	SDC	0.29	0.28	0.03
	non-SDC	3.96	3.68	2.55
2000	SDC	0.31	0.29	0.05
	non-SDC	72.96	65.55	52.13

4 Related Work and Publications

Very few existing works address the computational quality of Web service discovery techniques. I am not aware of any other approach that addresses this problem in a similar way. The following outlines the relationship to related research fields; details are discussed in the publications listed below.

Semantic Web Service Discovery. Most works are only concerned with the matchmaking techniques. As a contribution to this end, my work is based on a formal model that describes requested and provided functionalities on the level of executions of Web services and solutions for goals (*cf.* Section 2).

Web Service Repository Indexing. Other approaches reduce the search space for discovery by indexing Web service repositories. Keyword-based categorization as already supported by UDDI is imprecise in comparison to the SDC graph. More sophisticated solutions create a search tree based on formal descriptions; this can achieve logarithmic search time, but – in contrast to SDC – still requires several matchmaking operations for each request.

Caching. Caching techniques are a well-established means for performance increase in several areas of computing. Respective studies show that caching can achieve the highest efficiency increase if there are many similar requests. The SDC graph can be understood as a cache structure for Web service discovery.

Scalable Ontology Repositories. Works on scalable ontology reasoning infrastructures minimize the reasoning effort at runtime, e.g. by materalization and organization of the available knowledge at design time. However, such techniques can not replace the SDC technique because it defines a specific knowledge structure and algorithms for Web service discovery.

Publications (most relevant)

Stollberg, M. and Norton, B.: *A Refined Goal Model for Semantic Web Services.* In Proc. of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007), Mauritius, 2007.

Stollberg, M.; Keller, U.; Lausen, H. and Heymans, S.: *Two-phase Web Service Discovery based on Rich Functional Descriptions.* In Proc. of the 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Austria, 2007.

Stollberg, M.; Hepp, M., Hoffmann, J.: *Efficient and Scalable Web Service Discovery with Caching.* Submitted to 6th International Semantic Web Conference (ISWC 2007).