

Caching for Semantic Web Service Discovery

Michael Stollberg

Digital Enterprise Research Institute (DERI),
University of Innsbruck,
Technikerstrasse 21a, A-6020 Innsbruck, Austria
michael.stollberg@deri.org

Abstract. The presented PhD work develops a technique for increasing the computational efficiency of semantically enabled Web service discovery. This aims at enabling semantic SOA technologies to deal with the large numbers of available Web services in real-world settings. Adopting the concept of caching, the approach is to capture results of design time discovery and then effectively utilize this knowledge for runtime discovery. The work is evaluated by a time efficiency comparison with other Web service discovery engines, and by a statistical applicability evaluation in real-world SOA scenarios.

Keywords: Semantic Web Services, Goals, Discovery, Efficiency

1 Introduction

The telecommunication provider *Verizon* (www.verizon.com) maintains one of the world's largest SOA systems with more than 600 applications that use about 3.500 registered Web services. One of the central problems therein is **discovery**, i.e. to find the Web service that can be used for solving a specific problem in an application context. Works in the field of Semantic Web services aim at automation of Web service discovery by reasoning on semantic descriptions. Apart from the necessary matchmaking, such techniques must be able to deal with large numbers of Web services in order to stay operational in real-world scenarios.

My work addresses this by applying the concept of caching to Web service discovery. The basis for this are goals, i.e. formalized client objectives as 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.

This is a novel approach that exposes a well-established means for increasing computational efficiency to semantically enabled Web service discovery. Related works aim at reducing the computational costs by search over indexed Web service repositories. In contrast, my approach is based on the usability of Web services in an application context (i.e. for solving a goal). This re-focuses discovery from what a Web service can do towards which problems it can solve.

This paper explains the approach and positions it in related work (Sec. 2), outlines the planned evaluation (Sec. 3), and concludes with the status of the PhD work (Sec. 4). Details are addressed in the following publications: [3] discusses the problem statement and explains the research methodology, [6] presents the developed goal model, [5] presents the matchmaking technique for Web service discovery, and [4] provides a detailed specification of the caching technique.

2 Approach, Solution, and Related Work

2.1 Web Service Discovery Framework

My work is based on the Web service discovery framework defined for WSMO. This understands a *Web service* as a software artifact that is formally described and provides access to real-world *services*. A goal formally describes an objective that a client wants to solve by using Web services. I extend this with the differentiation of *goal templates* as abstract objective descriptions and *goal instances* that describe concrete client objectives by instantiating a goal template with particular inputs. Consider the objective of buying a ticket from Innsbruck to Vienna: the goal template formally describes this on the schema level (i.e. conditions on the origin and destination, the means of transport, etc., with respect to a background ontology), and the goal instance instantiates this with the concrete values (i.e. origin = Innsbruck, destination = Vienna, etc.).

Discovery is concerned with finding a Web service out of the available ones that can be used to solve a goal. I distinguish between **design time** and **runtime** Web service discovery as illustrated in Fig. 1. The matchmaking techniques are based on rich, formal descriptions of requested and provided functionalities [5].

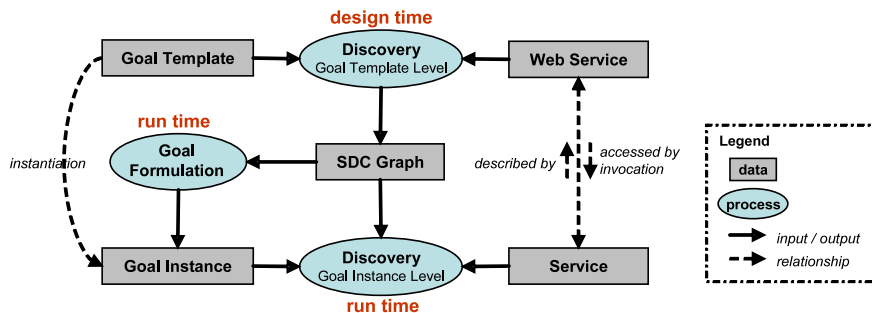


Fig. 1. SDC-enabled Web Service Discovery: Concepts and Operations

At design time – that is whenever a goal template or a Web service is added, modified, or removed – *discovery on the goal template level* is performed. This considers all aspects relevant for the usability of a Web service: *functional* discovery by matchmaking of the formally described requested and provided functionalities, and *non-functional* discovery on the quality-of-service aspects and behavioral compatibility. The result is stored in the *SDC graph*, the knowledge structure for enabling efficient Web service discovery (see below).

At runtime, a client defines a goal instance for which the usable Web services need to be determined. At first, *goal formulation* is concerned with specifying the client objective in terms of a goal. For this, the client chooses a goal template from the SDC graph and creates a goal instance thereof. This can be supported by graphical user interfaces; optionally, the chosen goal template is replaced by a more appropriate one. Then, the *discovery on the goal instance level* determines the usable Web services. This only needs to consider those Web services that are usable for the corresponding goal template (pre-filtering), and may omit matchmaking under certain matching degrees (minimal use of a reasoner) [5].

2.2 Semantic Discovery Caching

The main contribution of my work is the *Semantic Discovery Caching* technique, short: SDC. Its central element is the *SDC graph* that captures design time discovery results. This knowledge is effectively used by algorithms for both runtime and design time discovery. The complete specification is provided in [4].

The SDC graph consists of a *goal graph* that organizes goal templates in a subsumption hierarchy with respect to their semantic similarity, and a *discovery cache* that keeps minimal knowledge about the usability of the available Web services for each goal template. Two goal templates $\mathcal{G}_1, \mathcal{G}_2$ are considered to be *similar* if they have at least one common solution. This is expressed in terms of matching degrees between their formal functional descriptions (*exact, plugin, subsume, intersect, disjoint*); the same degrees are used to denote the functional usability of a Web service W for a goal template \mathcal{G} . Arcs in the SDC graph are directed connections of the form $d(\text{source}, \text{target})$: the source is always a goal template; arcs with a goal template as the target constitute the goal graph, and those with a Web service as the target define the discovery cache; d describes the matching degree between the source and the target. Formally, the SDC graph is set of *directed acyclic graphs* (DAG). In each connected sub-graph, the inner nodes are similar goal templates and the leaf nodes are the usable Web services.

For illustration, let us consider two goal templates: \mathcal{G}_1 for finding the best restaurant in an Austrian city, and \mathcal{G}_2 for finding the best Italian restaurant in a European city. Among others, let there be a Web service W_1 that finds the best restaurant in the any city of the world, and a W_5 that finds the best Italian restaurant in an Austrian city. The result of design time discovery is that both Web services are usable for each of the goal templates, under different matching degrees. This is kept in the SDC graph as shown in Fig. 2. To ensure that the SDC graph exposes the formal properties for efficient Web service discovery, several refinements are performed during its stepwise creation.

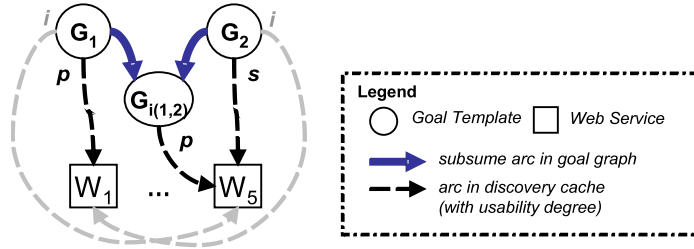


Fig. 2. Example of a SDC graph

The first refinement concerns the goal graph. In the example, the similarity degree between \mathcal{G}_1 and \mathcal{G}_2 is *intersect*: if there is an Austrian city wherein the best restaurant is Italian, then this is a solution for both \mathcal{G}_1 and \mathcal{G}_2 . Such an intersect arc does not provide much valuable information, and it may cause cycles in the goal graph that hamper its search properties. Thus, this is resolved by defining an *intersection goal template* that describes the common solutions and becomes a child of both original goal templates (cf. $\mathcal{G}_{i(1,2)}$ in Fig. 2). We do not store arcs under the *exact* or *disjoint* similarity degree, as these would not provide any valuable information. This results in the goal graph to only consist of subsume arcs (for the *plugin* degree the inverse arc is kept): $subsume(\mathcal{G}_i, \mathcal{G}_j)$ denotes that the set of solutions for a child node \mathcal{G}_j is a proper subset of the parent node \mathcal{G}_i . In consequence, the set of usable Web services for \mathcal{G}_j is a proper subset of those that are usable for \mathcal{G}_i .

The other refinement ensure the minimality of the discovery cache. The first one is the omittance of directly inferrable arcs: if a Web service W is usable for a parent node \mathcal{G}_i under the *exact* or *plugin* degree, then the usability degree of W for a child node \mathcal{G}_j is always *plugin*. Thus, the arc between \mathcal{G}_j and W is omitted. In the example, such an omitted arc is the one between W_1 for \mathcal{G}_2 . Its value would be a *plugin* usability degree; this can be directly inferred from the *plugin* degree between the W_1 and the parent node \mathcal{G}_1 . Secondly, only the arc with the highest usability degree is kept. This is based on the preference order $exact, plugin > subsume > intersect$ that reflects the need for matchmaking at runtime discovery: if W is usable for a goal template \mathcal{G} under either the *exact* or the *plugin*, then it is also usable for each goal instance that instantiates \mathcal{G} ; the *subsume* and the *intersect* degree require matchmaking for determining this. Finally, an external ranking function defines a priority order of the usable Web services for a goal template on the basis of non-functional discovery.

The SDC graph is used to minimize the computational costs for Web service discovery by efficient pre-filtering and minimal use of the matchmaker. Consider a goal instance that requests to find the best restaurant in Innsbruck (an Austrian city); let us assume that this is of type Italian by accident. Hence, this is an instantiation of the intersection goal template $\mathcal{G}_{i(1,2)}$. The usability degree for of both W_1 and W_5 for $\mathcal{G}_{i(1,2)}$ is *plugin* degree, and thus both are usable for any of it instantiations. Assuming a priority ranking $W_5 > W_1$, we first try to

solve the goal instance with W_5 and – in case of some execution failure – we use W_1 . Moreover, the knowledge in the SDC graph can also be used at design time discovery. Due to the subsumption hierarchy in the goal graph, here only those Web service that are usable for either \mathcal{G}_1 or \mathcal{G}_2 can be usable for $\mathcal{G}_{i(1,2)}$.

2.3 Related Work

The need for scalable reasoning techniques is a commonly known requirement for the leverage of Semantic Web technologies. The same arises for semantic SOA technologies in order to stay reliable in real-world settings. I am not aware of any other approach that addresses this problem in the way outlined above. The following briefly discusses related works and positions my approach therein.

Semantic Web Service Discovery. Most works are concerned with the necessary semantic matchmaking and the underlying descriptions of Web services and goals. As a contribution to this end, my work is based on a rich model that describes requested and provided functionalities on the level of executions of Web services and solutions for goals. Defined in a first-order logic framework, this allows to achieve more precise functional discovery results (*cf.* [5] for details).

Web Service Indexing and Pre-Filtering. The reduction of the search space for Web service discovery is already addressed in UDDI. Therein, a keyword-based categorization is used to pre-filter possibly usable candidates. In comparison to the SDC graph, this categorization is imprecise with respect to the usability of a Web service – even if the keywords are based on an ontology. Another approach is to index available Web services by creating a search tree with respect to the formally described requested functionalities (e.g. [1]). Although this may achieve a logarithmic search time (if the tree is balanced), still matchmaking is required for each new incoming request. In contrast, the SDC technique can – in certain cases – detect a usable Web service for a goal instance by lookup.

Caching. As a well-established means for performance increase, caching techniques are applied in several areas of computing. Adopted in SDC, the idea is to answer requests from pre-computed results. Respective studies show that caching can achieve the highest efficiency increase if there are many semantically similar requests. For SDC, this is subject of the evaluation (see below).

Scalable Ontology Repositories. Works on scalable ontology reasoning infrastructures minimize the reasoning effort at runtime by pre-computing and organization of the available knowledge at design time (e.g. [2]). Such optimized ontology repositories can not replace the SDC technique because it defines the specific knowledge structure and algorithms for Web service discovery.

3 Evaluation

A proper evaluation requires two aspects: a prototype implementation for demonstration and comparison with other discovery engines, and an applicability study with respect to the specifics of the SDC technique. Both are ongoing, respectively future aspects of my PhD work.

The prototype is implemented as a discovery component in the WSMX system (the WSMO reference implementation, see www.wsmx.org). This shall be integrated into the overall system architecture in order to make use of existing components for functional and non-functional discovery as well as for data mediation. The achievable efficiency increase will be demonstrated by a comparison of the SDC-enabled Web service discovery with other discovery engines with the shipment scenario from the SWS Challenge (http://sws-challenge.org/wiki/index.php/Scenario:_Shipment_Discovery).

The applicability study will evaluate whether the design of the SDC technique is suitable for real-world settings. As a caching technique, it is dependent on the structure of the application context. The optimal situation is if there are many goal templates that can be organized in a fine grained goal graph (for pre-filtering), and if there are many Web services that are usable under the *exact* or *plugin* degree (for omittance of matchmaking). The working hypothesis is that this correlates to the most common situation in real-world SOA applications. This shall be verified by a empirical study of existing SOA systems (e.g. at Verizon, see above) as well as designated use cases for Semantic Web services.

4 Status and Future Work

I am in the third year of my PhD studies; however, due to internal reallocations I only commenced with the presented PhD work about 1.5 years ago.

At the point of writing, the specification of the extended discovery framework as well as of the SDC technique is completed. The evaluation outlined in Section 3 is ongoing work. In the future, I will have to more exhaustively elaborate on related work and position my research therein, and then finally I will write up the thesis document.

References

1. I. Constantinescu, W. Binder, and B. Faltings. Flexible and Efficient Matchmaking and Ranking in Service Directories. In *Proc. of the 3rd International Conference on Web Services (ICWS 2005), Florida, USA, 2005*.
2. I. Horrocks, L. Li, D. Turi, and S. Bechhofer. The Instance Store: Description Logic Reasoning with Large Numbers of Individuals. In *Proc. of International Workshop on Description Logics (DL 2004), British Columbia, Canada, 2004*.
3. M. Stollberg. Semantic Caching for Service Oriented Architectures. In *Service-ware – DERI PhD Seminar, November, 2006*. online: <http://members.deri.at/~michaels/publications/mstollberg-PhDthesis0verview.pdf>.
4. M. Stollberg. Semantic Discovery Caching: Specification. Technical Report DERI-2007-02-03, DERI, 2007.
5. M. Stollberg, U. Keller, H. Lausen, and S. Heymans. Two-phase Web Service Discovery based on Rich Functional Descriptions. In *4th European Semantic Web Conference (ESWC 2007), 2007*. (submitted).
6. M. Stollberg and B. Norton. A Refined Goal Model for Semantic Web Services. In *In Proc. of the 2nd International Conference on Internet and Web Applications and Services (ICIW 2007), Mauritius, 2007*.