A Caching Mechanism for Semantic Web Service Discovery

Michael Stollberg, Martin Hepp, and Jörg Hoffmann

Digital Enterprise Research Institute (DERI),
University of Innsbruck, Austria
{firstname.lastname}@deri.org

Abstract. The discovery of suitable Web services for a given task is one of the central operations in Service-oriented Architectures (SOA), and research on Semantic Web services (SWS) aims at automating this step. For the large amount of available Web services that can be expected in real-world settings, the computational costs of automated discovery based on semantic matchmaking become important. To make a discovery engine a reliable software component, we must thus aim at minimizing both the mean and the variance of the duration of the discovery task. For this, we present an extension for discovery engines in SWS environments that exploits structural knowledge and previous discovery results for reducing the search space of consequent discovery operations. Our prototype implementation shows significant improvements when applied to the Stanford SWS Challenge scenario and dataset.

1 Introduction

Web service discovery is the process of finding suitable Web services for a given task, denoting one of the central operations in Service-oriented Architectures (SOA). There is already substantial work in the field Semantic Web services (SWS) on automating Web service discovery by semantic matchmaking, mostly focussing on the retrieval performance (e.g. [14,11,13,9]).

However, the computational performance of semantically enabled discovery and the practical consequences for SWS environments have not received a lot of attention so far. Considering the increasing amount of available Web services, this becomes in particular relevant for employing a discovery engine as a heavily used component in systems for dynamic Web service composition or semantically enabled business process management (e.g. [19,8]). For this, we consider the following characteristics for judging the computational reliability of a discovery engine: *efficiency* as the time required for finding a suitable Web service, *scalability* as the ability to deal with a large search space of available Web services, and *stability* as a low variance of the execution time of several invocations.

This paper presents a technique that addresses this challenge by adapting the concept of caching to Web service discovery. It captures knowledge on discovery results for generic descriptions of objectives to be achieved, and exploits this for optimizing Web service discovery for concrete requests at runtime.

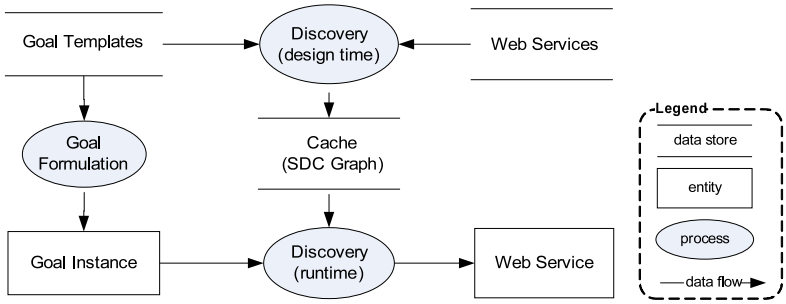

Fig. 1. Overview of Web Service Discovery Framework

We refer to this as *Semantic Discovery Caching* (SDC). Figure 1 shows the basic idea as a dataflow diagram. There are three core conceptual entities: *Web services* that have a formal description, *goal templates* as generic objective descriptions that are stored in the system, and *goal instances* that describe concrete requests by instantiating a goal template with concrete inputs. At design time, Web services for goal templates are discovered. The result is captured in a special knowledge structure called the *SDC graph*. At runtime, a concrete client request is formulated as a goal instance for which suitable Web services need to be discovered. As the expectably most frequent operation in SOA applications, we optimize this by exploiting the SDC graph in order to reduce the search space and minimize the number of necessary matchmaking operations.

The paper is structured as follows. Section 2 briefly describes our approach for semantically enabled Web service discovery as presented in earlier works [18,17]. Then, Section 3 specifies the Semantic Discovery Caching technique. Section 4 presents the evaluation, and discusses the relevance and specifics of our approach. Section 5 positions it within related work, and Section 6 concludes the paper. We use the shipment scenario from the Stanford SWS Challenge as a running example, a widely recognized initiative for demonstration and comparison of semantically enabled discovery techniques (`http://www.sws-challenge.org`). A detailed technical report on this work is provided in [16].

2 Foundations of the Discovery Framework

As the basis for the subsequent sections, the following summarizes our approach for semantically enabled Web service discovery as presented in earlier works.

We follow the goal-driven approach for Semantic Web services as promoted by the WSMO framework [5]. In contrast to an invocation request for a Web service, a goal describes a client objective with respect to the problem that shall be solved while abstracting from technical details on Web service invocation. The aim is to facilitate problem-oriented Web service usage: the client merely specifies the objective to be achieved as a goal, and the system automatically discovers, composes, and executes suitable Web services for solving this [18].

The distinction of goal templates and goal instances allows to ease the goal formulation by clients, and it facilities the two-phase Web service discovery as outlined above. For this, we have defined a formal model that considers a state-based model of the world that Web services operate in, and provides precise definitions of goals, Web services, and the necessary matchmaking techniques. We here recall the central aspects; the full model is defined in [17].

2.1 Web Services, Goals, and Functional Descriptions

We 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. The relevant parts of goal and Web service descriptions for discovery by semantic matchmaking are the formally described requested and the provided functionalities.

In our state-based model, a particular execution of a Web service W denotes a sequence of states $\tau = (s_0, \ldots, s_m)$, i.e. a change of the world from a start state s_0 to an end state s_m. In consequence, the overall functionality provided by W is the set of all possible executions of W, denoted by $\{\tau\}_W$. Analogously, we understand a particular solution of a goal as a sequence of states from the initial state into a state of the world wherein the objective is solved. A functional description $\mathcal{D}$ formally describes the possible executions of a Web service – respectively the possible solutions for a goal – with respect to the start- and end states.

We define $\mathcal{D}$ over a signature Σ, and use ontologies Ω as the background knowledge. $\mathcal{D}$ consists of a set of input variables $IN = \{i_1, \ldots, i_n\}$, a precondition ϕ^{pre} that constrains the possible start states, and an effect ϕ^{eff} that constrains the possible end states. To properly describe the dependency of the start- and end states, IN occur as free variables in both ϕ^{pre} and ϕ^{eff}; the predicate *out* denotes the outputs. The formal meaning of $\mathcal{D}$ is defined as an *implication semantics* between the precondition and the effect. We say that a Web service W provides the functionality described by $\mathcal{D}$, denoted by $W \models \mathcal{D}$, if and only if for all $\tau \in \{\tau\}_W$ holds that if $s_0 \models \phi^{pre}$ then $s_m \models \phi^{\mathit{eff}}$. In order to deal with functional descriptions in terms of model-theoretic semantics, we present this as a FOL formula $\phi^{\mathcal{D}}$ of the form $\phi^{pre} \Rightarrow \phi^{\mathit{eff}}$. Then, $W \models \mathcal{D}$ is given if and only if every $\tau \in \{\tau\}_W$ is represented by a Σ-interpretation that is a model of $\phi^{\mathcal{D}}$.

Analogously, the functional description $\mathcal{D}_G$ of a goal template G formally describes the set $\{\tau\}_G$ as the state sequences that are possible solutions for G. Goal templates are generic objective descriptions that are kept in the system. At runtime, a concrete client request is formulated as a goal instance that instantiates a goal template with concrete inputs. We define a goal instance as a pair $GI(G) = (G, \beta)$ with G as the corresponding goal template, and an input binding $\beta : \{i_1, \ldots, i_n\} \rightarrow \mathcal{U}$ as a total function that assigns objects of $\mathcal{U}$ to each IN-variable of $\mathcal{D}_G$. This β is subsequently used to invoke a Web service in order to solve $GI(G)$. We say that $GI(G)$ is a consistent instantiation of its corresponding goal template, denoted by $GI(G) \models G$, if $\phi^{\mathcal{D}_G}$ is satisfiable under the input binding β. A usable Web service for $GI(G)$ can only be found if this is given. Moreover, it holds that if $GI(G) \models G$ then $\{\tau\}_{GI(G)} \subseteq \{\tau\}_G$.

Table 1. Examples for Functional Descriptions of Goals and Web Services

Goal Template G "ship a package of any weight in Europe"	**Web Service** W "shipment in Germany, max 50 kg"
Ω: location & shipment ontology IN: {?s,?r,?p,?w} ϕ^{pre}: $person(?s) \wedge in(?s, europe)$ $\wedge\, person(?r) \wedge in(?r, europe)$ $\wedge\, package(?p) \;\wedge weight(?p, ?w)$ $\wedge\, maxWeight(?w, heavy)$. ϕ^{eff}: $\forall ?o, ?price.\ out(?o) \Leftrightarrow ($ $shipmentOrder(?o, ?p)$ $\wedge\, sender(?p, ?s) \;\wedge receiver(?p, ?r)$ $\wedge\, costs(?o, ?price)\ )$.	Ω: location & shipment ontology IN: {?s,?r,?p,?w} ϕ^{pre}: $person(?s) \wedge in(?s, germany)$ $\wedge\, person(?r) \wedge in(?r, germany)$ $\wedge\, package(?p) \;\wedge weight(?p, ?w)$ $\wedge\, maxWeight(?w, 50)$. ϕ^{eff}: $\forall ?o, ?price.\ out(?o) \Leftrightarrow ($ $shipmentOrder(?o, ?p)$ $\wedge\, sender(?p, ?s) \;\wedge receiver(?p, ?r)$ $\wedge\, costs(?o, ?price)\ )$.

Table 1 shows examples for functional descriptions in our running example. Using classical first-order logic as the specification language, the preconditions define conditions on the required inputs, and the effects state that the output is a shipment order with respect to the input values.[1]

2.2 Semantic Matchmaking

The matchmaking techniques for Web service discovery are defined on the basis of the functional descriptions explained above. We consider a Web service W to be usable for a goal template G if there exists at least one execution of W that is also a solution for G, i.e. if $\exists \tau.\ \tau \in \{\tau\}_G \cap \{\tau\}_W$. We express the usability of W for G in terms of matching degrees as defined in Table 2. Four degrees – *exact, plugin, subsume, intersect* – denote different situations wherein W is usable for solving G; the *disjoint* degree denotes that this is not given. We always use the highest possible degree as it holds that (1) *plugin* $\wedge$ *subsume* $\Leftrightarrow$ *exact*, (2) *plugin* $\Rightarrow$ *intersect*, (3) *subsume* $\Rightarrow$ *intersect*, and (4) $\neg$ *intersect* $\Leftrightarrow$ *disjoint*.

Analogously, we consider a Web service W to be usable for a goal instance $GI(G)$ if W can provide a solution for $GI(G)$ when it is invoked with the input binding β defined in $GI(G)$. Formally, this is given if union of the formulae $\Omega \cup \{[\phi^{\mathcal{D}_G}]_\beta, [\phi^{\mathcal{D}_W}]_\beta\}$ is satisfiable. This means that under consideration of the domain knowledge Ω and under the input binding β defined in $GI(G)$ there must be a Σ-interpretation that represents a solution for the corresponding goal template G as well as a possible execution of the Web service W. However, we

[1] (a) We consider Web services to provide *deterministic functionalities*, i.e. that the end-state of an execution is completely dependent on the start-state and the provided inputs; this is a pre-requisite for precise discovery by semantic matchmaking. (b) We consider all functional descriptions $\mathcal{D}$ to be *consistent*, i.e. that $\phi^{\mathcal{D}}$ is satisfiable under an input binding β. Otherwise, a Web service $W \models \mathcal{D}$ would not realizable, and there would not be any solution for a goal. The full model further considers dynamic symbols that are changed during executions.

Table 2. Definition of Matching Degrees for $\mathcal{D}_G$, $\mathcal{D}_W$

Denotation	Definition	Meaning
exact$(\mathcal{D}_G, \mathcal{D}_W)$	$\Omega \models \forall\beta.\ \phi^{\mathcal{D}_G} \Leftrightarrow \phi^{\mathcal{D}_W}$	$\tau \in \{\tau\}_G$ if and only if $\tau \in \{\tau\}_W$
plugin$(\mathcal{D}_G, \mathcal{D}_W)$	$\Omega \models \forall\beta.\ \phi^{\mathcal{D}_G} \Rightarrow \phi^{\mathcal{D}_W}$	if $\tau \in \{\tau\}_G$ then $\tau \in \{\tau\}_W$
subsume$(\mathcal{D}_G, \mathcal{D}_W)$	$\Omega \models \forall\beta.\ \phi^{\mathcal{D}_G} \Leftarrow \phi^{\mathcal{D}_W}$	if $\tau \in \{\tau\}_W$ then $\tau \in \{\tau\}_G$
intersect$(\mathcal{D}_G, \mathcal{D}_W)$	$\Omega \models \exists\beta.\ \phi^{\mathcal{D}_G} \wedge \phi^{\mathcal{D}_W}$	there is a τ such that $\tau \in \{\tau\}_G$ and $\tau \in \{\tau\}_W$
disjoint$(\mathcal{D}_G, \mathcal{D}_W)$	$\Omega \models \neg\exists\beta.\ \phi^{\mathcal{D}_G} \wedge \phi^{\mathcal{D}_W}$	there is no τ such that $\tau \in \{\tau\}_G$ and $\tau \in \{\tau\}_W$

can simplify the determination of the usability of W for $GI(G)$ on the basis of the usability degree of W for the corresponding goal template G as follows.

Definition 1. *Let $GI(G) = (G, \beta)$ be a goal instance with $GI(G) \models G$. Let W be a Web service, and let $\mathcal{D}_W$ be a functional description such that $W \models \mathcal{D}_W$. W is usable for solving $GI(G)$ if and only if:*

(i) $\texttt{exact}(\mathcal{D}_G, \mathcal{D}_W)$ *or*
(ii) $\texttt{plugin}(\mathcal{D}_G, \mathcal{D}_W)$ *or*
(iii) $\texttt{subsume}(\mathcal{D}_G, \mathcal{D}_W)$ *and $\Omega \wedge [\phi^{\mathcal{D}_W}]_\beta$ is satisfiable, or*
(iv) $\texttt{intersect}(\mathcal{D}_G, \mathcal{D}_W)$ *and $\Omega \wedge [\phi^{\mathcal{D}_G}]_\beta \wedge [\phi^{\mathcal{D}_W}]_\beta$ is satisfiable.*

This states that only those Web services that are usable for the corresponding goal template G are potentially usable for the goal instance $GI(G)$. If a Web service W is usable for G under the *exact* or *plugin* degree, then it is also usable for any goal instance of G because $\{\tau\}_{GI(G)} \subseteq \{\tau\}_G \subseteq \{\tau\}_W$. Under the *subsume* degree, all executions of W are solutions of G but not vice versa. Table 1 above is an example for this. Consider a goal instance that defines $\beta = \{?s|paris, ?r|vienna, ?p|aPackage, ?w|3.1\}$: although this properly instantiates the goal template, β does not allow to invoke the Web service (which is restricted to Germany); thus, it is not usable here. Under the *intersect* degree, the complete matching condition explained above must be checked at runtime.

3 Semantic Discovery Caching

We now turn towards the caching mechanism for Web service discovery. Working on the formal model explained above, the aim is to improve the computational reliability of Web service discovery for goal instances that is performed at runtime. We commence with the design principles, then provide the formal definition, and finally explain the optimization for the runtime discovery process.

3.1 Overview

The idea is to reduce the search space and minimize the necessary matchmaking operations for Web service discovery by exploiting the formal relationships between goal templates, goal instances, and Web services. The central element for

this is the SDC graph that organizes goal templates in a subsumption hierarchy with respect to their semantic similarity, and captures the minimal knowledge on the functional usability of the available Web services for the goal templates.

Two goal templates G_i and G_j are considered to be similar if they have at least one common solution. Then, mostly the same Web services are usable for them. We express this in terms of *similarity degrees* $d(G_i, G_j)$ that denote the matching degree between the functional descriptions $\mathcal{D}_{G_i}$ and $\mathcal{D}_{G_j}$. Formally, these degrees are defined analog to Table 2 (*cf.* Section 2.2). In order to enable efficient search, we define the SDC graph such that the only occurring similarity degree is $subsume(G_i, G_j)$. If this is given, then (1) the solutions for the child G_j are a subset of those for the parent G_i, and thus (2) the Web services that are usable for G_j are a subset of those usable for G_i.

In consequence, the SDC graph is a directed acyclic graph that consists of two layers. The upper one is the *goal graph* that defines the subsumption hierarchy of goal templates by directed arcs. The lower layer is the *usability cache* that explicates the usability of each available Web service W for every goal template G by directed arcs that are annotated with the usability degree $d(G, W)$. The discovery operations use this knowledge structure by inference rules of the form $d(G_i, G_j) \wedge d(G_i, W) \Rightarrow d(G_j, W)$ that result from the formal definitions.

Figure 2 illustrates the SDC graph for our running example along with the most relevant inference rules. There are three goal templates: G_1 for package shipment in Europe, G_2 for Switzerland, and G_3 for Germany. Their similarity degrees are $subsume(G_1, G_2)$ and $subsume(G_1, G_3)$, which is explicated in the goal graph. Consider some Web services, e.g. W_1 for package shipment in Europe, W_2 in the whole world, W_3 in the European Union, and W_4 in the Commonwealth. Their usability degree for each goal template is explicated in the usability cache, whereby redundant arcs are omitted. We shall explain the creation of the SDC graph as well as its usage for optimizing the discovery process below.

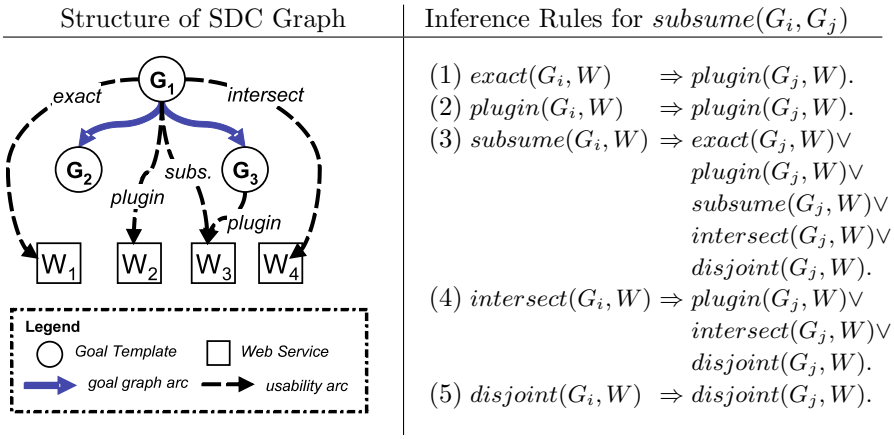

Fig. 2. Example of a SDC Graph and Inference Rules

3.2 Definition

The following provides the formal definition of the SDC graph and explains the algorithms for ensuring that the properties are maintained at all times.

Definition 2. *Let $d(G_i, G_j)$ denote the similarity degree of goal templates G_i and G_j, and let $d(G, W)$ denote the usability degree of a Web service W for a goal template G. Given a set $\mathcal{G}$ of goal templates and a set $\mathcal{W}$ of Web services, the SDC graph is a directed acyclic graph $(V_\mathcal{G} \cup V_\mathcal{W}, E_{sim} \cup E_{use})$ such that:*

(i) $V_\mathcal{G} := \mathcal{G} \cup \mathcal{G}^I$ *is the set of inner vertices where:*
- $\mathcal{G} = \{G_1, \ldots, G_n\}$ *are the goal templates; and*
- $\mathcal{G}^I := \{G^I \mid G_i, G_j \in \mathcal{G}, d(G_i, G_j) = \mathsf{intersect}, G^I = G_i \cap G_j\}$ *is the set of intersected goal templates from* $\mathcal{G}$

(ii) $V_\mathcal{W} := \{W_1, \ldots, W_m\}$ *is the set of leaf vertices representing Web services*

(iii) $E_{sim} := \{(G_i, G_j) \mid G_i, G_j \in V_\mathcal{G}\}$ *is the set of directed arcs where:*
- $d(G_i, G_j) = \mathsf{subsume}$*; and*
- *not exists* $G \in V_\mathcal{G}$ *s.t.* $d(G_i, G) = \mathsf{subsume}$, $d(G, G_j) = \mathsf{subsume}$.

(iv) $E_{use} := \{(G, W) \mid G \in V_\mathcal{G}, W \in V_\mathcal{W}\}$ *is set of directed arcs where:*
- $d(G, W) \in \{\mathsf{exact}, \mathsf{plugin}, \mathsf{subsume}, \mathsf{intersect}\}$*; and*
- *not exists* $G_i \in V_\mathcal{G}$ *s.t.* $d(G_i, G) = \mathsf{subsume}$, $d(G_i, W) \in \{\mathsf{exact}, \mathsf{plugin}\}$.

This defines the structure of a SDC graph as outlined above. Two refinements are necessary to obtain this from an initial set of goal templates and Web services.

The first one ensures that the only similarity degree that occurs in the SDC graph is $subsume(G_i, G_j)$, *cf.* clause (iii). This denotes that G_j is a functional specialization of G_i such that $\{\tau\}_{G_j} \subset \{\tau\}_{G_i}$. In consequence, only those Web services that are usable for G_i can be usable for G_j because if $\{\tau\}_{G_i} \cap \{\tau\}_W = \emptyset$ then also $\{\tau\}_{G_j} \cap \{\tau\}_W = \emptyset$, *cf.* rule (5) in Figure 2. With this as its constituting element, the SDC graph provides an index structure for efficient search of goal templates and Web services as explained above. The other possible similarity degrees are handled as follows: if $exact(G_i, G_j)$, only one goal template is kept while the other one is redundant; if $plugin(G_i, G_j)$ then we store the opponent arc (G_j, G_i). If $disjoint(G_i, G_j)$, then both are kept as disconnected nodes in the SDC graph. Effectively, each of its connected subgraphs covers a problem domain, e.g. one for the shipment scenario and another one for flight ticketing.

The only critical similarity degree is $intersect(G_i, G_j)$, denoting that G_i and G_j have a common solution but there are also exclusive solutions for each. This can cause cycles in the SDC graph which hamper its search properties. To avoid this, we create an *intersection goal template* $G^I(G_i, G_j)$ whose solutions are exactly those that are common to G_i and G_j, *cf.* clause (i). Formally, G^I is defined as the conjunction of the functional descriptions of the original goal templates, i.e. $\phi^{\mathcal{D}_{G^I(G_i,G_j)}} = \phi^{\mathcal{D}_{G_i}} \wedge \phi^{\mathcal{D}_{G_j}}$ so that $\{\tau\}_{G^I(G_i,G_j)} = \{\tau\}_{G_i} \cap \{\tau\}_{G_j}$. Because of this, it holds that $subsume(G_i, G^I(G_i, G_j))$ and $subsume(G_j, G^I(G_i, G_j))$. Thus, G^I becomes a child node of both G_i and G_j in the goal graph. This is applied for every occurring *intersect* similarity degree so that eventually all similar goal templates are organized in a subsumption hierarchy and no cycles occur in

the SDC graph. Intersection goal templates are only used as logical constructs; their functional descriptions do not have to be materialized.

The second refinement ensures the minimality of the usability cache, *cf.* clause (iv). For optimizing the discovery operations, we must know the usability degree of every Web service for each goal template. However, in order to avoid redundancy, we omit arcs for which the precise usability degree can be inferred from the SDC graph. It holds that if $subsume(G_i, G_j)$, then the usability degree of a Web service W for the child G_j is always *plugin* if W is usable for the parent G_i under the degrees *exact* or *plugin* because $\{\tau\}_W \supseteq \{\tau\}_{G_i} \supset \{\tau\}_{G_j}$. Thus, the arc (G_j, W) is not explicated in the SDC graph. In the above example, the Web services W_1 and W_2 are usable under the *plugin* degree for both G_2 and G_3; this can be inferred from the usability cache arcs of G_1 (*cf.* rules (1) and (2) in Figure 2). Therewith, E_{use} is the minimal set of arcs that are necessary to explicate the usability degrees of the available Web services for each goal template.

In our implementation, the creation of a SDC graph is realized by the subsequent addition of goal templates. Applying the refinements explained above, a new goal template is first inserted at the right position in the goal graph and then the usability cache is created for it. The removal or modification of goal templates are manual maintenance operations; respective algorithms are ensure that the properties of the SDC graph are maintained. Analogous algorithms are provided for the addition, removal, and modification of Web services. These are automatically triggered by changes in the Web service repository.[2]

3.3 Runtime Discovery Optimization

We now explain the usage of the SDC graph for optimizing the runtime discovery process, i.e. for finding a Web service that is usable for solving a goal instance. We consider this as the most frequent operation in real-world SOA applications, while changes on goal templates and Web services are significantly less frequent maintenance operations. The optimization is achieved by (1) *reducing the search space* as only the Web services that are usable the corresponding goal template need to be inspected, and (2) *minimizing the number of necessary matchmaking operations* by first inspecting Web services for which no matchmaking is required at runtime. Listing 1 illustrates the algorithm for this.

```
input: GI(G);
  if ( ! consistentInstantiation (GI(G)) ) then fail ;
  if ( lookup(G) ) then return W;
  while ( subsume(G,G') and consistentInstantiation (GI(G')) ) do {
     replace(G,G');
     if ( lookup(G') ) then return W; }
  if ( otherWS(G) ) then return W;
   else fail ;
```

Listing 1. Algorithm for SDC-enabled Runtime Discovery

[2] The SDC prototype is open source software available from the SDC homepage at `members.deri.at/~michaels/software/sdc/`. It is realized as a discovery component in the WSMX system (the WSMO reference implementation, `www.wsmx.org`). We use VAMPIRE for matchmaking, a FOL automated theorem prover.

The input is a goal instance $GI(G) = (G, \beta)$ for which a usable Web service shall be found. At first, we need to ensure that this is a consistent instantiation of its corresponding goal template G; if this is not given, a usable Web service can not be found. Then, we try to find a usable Web service by *lookup*. This searches in the SDC graph for a Web service W that is usable for G under the *exact* or the *plugin* degree; this W is usable for solving $GI(G)$ without the need of matchmaking at runtime (*cf.* Definition 1, Section 2.2). If this is successful, W is returned and the discovery is completed successfully.

Otherwise, we continue with refining the goal instance in order to reduce the search space. For this, we successively replace the corresponding goal template G by the child node G' for which the goal instance still is a consistent instantiation. In the example from Figure 2, let $GI(G_1)$ be a goal instance for shipping a package from Munich to Berlin that instantiates G_1 for package shipment within Europe. This is also a proper instantiation of G_3 for shipment within Germany; hence, we refine the goal instance to $GI(G_3)$. In the SDC Graph, all children of G are disjoint – those for which there is no intersection goal template – so that there can only be one G' with $subsume(G, G')$ and $GI(G) \models G'$. If there is an intersection goal template G^I and $GI(G) \models G^I$, this is found by following the path via either of its parents. We thus can search downwards in the goal graph until finding the lowest possible G': for this, the number of usable Web services is minimal. In each refinement step we invoke the *lookup* procedure because the probability of success is the higher the lower G' is allocated in the goal graph.

As the last option for finding a usable Web service, we inspect those ones that are usable for the (possibly refined) corresponding goal template under the degrees *subsume* and *intersect*; this requires matchmaking at runtime (*cf.* Definition 1). As soon as a usable Web service is detected, it is returned as the discovery result Otherwise, a Web service for solving $GI(G)$ does not exist.

This algorithm finds one Web service that is usable for a goal instance under functional aspects. Of course other aspects are also relevant for discovery. Most prominent in literature are *selection* and *ranking* techniques: the former reduces the set of suitable Web services with respect to quality-of-service criteria (e.g. [20]); the latter provides a preference order for the Web services (e.g. [12]). Under the reasonable premise of performing selection and ranking *after* functional discovery, the SDC technique provides a sufficient optimization technique for integrated discovery engines: the bottleneck is the number of available Web services, which is only relevant for the first processing step. The above algorithm can easily be modified to return the set of all functionally usable Web services.

4 Evaluation

In order to evaluate the performance gain achievable with the SDC technique, we have compared the our prototype implementation with a not-optimized discovery engine for goal instances that applies the same matchmaking techniques. The following explains the test set-up and methodology, summarizes the results, and discusses the impact of our observations.

4.1 Methodology

The aim of this evaluation is to quantify the effect of the SDC technique on the duration of realistic discovery tasks over larger sets of available Web services that can be expected in real-world settings. We therefore compare the SDC-enabled runtime discovery with a naive discovery engine for goal instances. We will discuss the relationship to other optimization techniques in Section 5.

For the comparison, we use the original data set from the Stanford SWS challenge shipment scenario that already served as the running example above. Based on real-world services, this challenge defines five Web services for package shipment from the USA to different destination countries, and several examples of client requests. We map this to our framework such that goal templates are generic objective descriptions for package shipment, and the individual requests are described as goal instances. The formal functional descriptions of goals and Web services are analog to Table 1, *cf.* Section 2.1. The root goal template of the resulting SDC graph describes the objective of shipping a package of any weight from and to anywhere in the world. The more detailed levels of the goal graph are concerned with shipment between continents, the next levels between countries, and the lowest levels differentiate the weight classes. At the top of the goal graph, the most common usability degree for the Web services is *subsume*; this changes to *plugin* at the lower levels.

On this basis, we define ten goal instances for which a usable Web service is to be found. These are modeled such that each part of the SDC-enabled runtime discovery algorithm is covered (*cf.* Section 3.3). The comparison engine is a naive runtime discoverer that does not apply any optimization techniques. It retrieves the available Web services in a random order, and performs the basic matchmaking to determine their usability for a goal instance as defined in Section 2.2. It uses the same matchmaking techniques and infrastructure as the SDC-enabled engine. For comparing the behavior of the engines, we perform Web service discovery for each goal instance with different numbers of available Web services. Among these are always the five Web services defined in the scenario that are potentially usable for the tested goal instance; all others are not.

4.2 Results

For the analysis, each comparison test has been run 50 times and the results are prepared in the following statistical standard notations: the *arithmetic mean* μ as the average value, the *median* $\bar{x}$ that denotes the value in the middle of the 50 test runs, and the standard deviation σ as a measurement for the value spread among the test runs. Table 3 shows a fragment of the augmented data of all test runs for all ten goal instances; the original data is provided in [16].[3]

[3] The comparison has been performed as a JUnit test for Java 5.0 in Eclipse 3.2; the test machine was a standard laptop with a 2 GHz Intel processor and 1 GB of RAM. For this use case, the average time for a single matchmaking operation has been 115 msec, and 15 msec for the discovery-by-lookup procedure.

Table 3. 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 (11.74 %)
	naive	0.41	0.39	0.21 (51.71 %)
100	SDC	0.29	0.28	0.03 (11.53 %)
	naive	3.96	3.68	2.55 (64.48 %)
1000	SDC	0.29	0.29	0.04 (14.79 %)
	naive	37.69	33.22	26.28 (69.70 %)
2000	SDC	0.31	0.29	0.05 (18.03 %)
	naive	72.96	65.55	52.13 (71.45 %)

From this we can observe the following differences between the compared engines with respect to the three quality criteria for reliability: the SDC-enabled discovery is in average faster than the naive engine, even for smaller numbers of Web services (*efficiency*); the time required by the SDC-engine is independent of the number of available Web services while it grows proportionally for the naive engine (*scalability*); over several invocations, the SDC-engine varies a lot less than the naive engine (*stability*). The high variation of the naive engine results from the randomized order under which the available Web services are examined. In this particular use case, the SDC optimization is mainly achieved by the pre-filtering via goal templates; the refinement step in the discovery algorithm reveals its potential when there are more usable Web services.

This indicates that the SDC technique helps to satisfy the requirements for using a Web service discovery engine as a reliable component in large SOA systems. Future-oriented works for Web service composition or business process management envision that the actual Web services for execution are dynamically discovered at runtime in order to provide better flexibility and allow compensation (e.g. [19,8]). Considering that compositions or processes can be complex and may consist of several Web services, the absolute overhead and the predictability of the discovery engine becomes a pre-requisite for realizing such technologies.

4.3 Discussion

For the applicability of the SDC technique in real-world applications not only the performance but also the appropriateness of the conceptual model is relevant.

To verify the our approach in this respect, we have examined its applicability for the SOA system maintained by the US-based telecommunication provider *Verizon*. This contains nearly 4000 Web services that provide basic functionalities for managing customer, product and order information; they differ in the specific usage conditions (e.g. in- and outputs, data formats, etc.). These are used by more than 600 applications that integrate the Web services into their software. The main challenges reported by the system providers is the precision of the discovery technique and the management of changes in the system. The former

can be increased by discovery techniques with semantic matchmaking. The latter results from the currently hard-wired invocation of the Web services by the client applications. To overcome this, the usage tasks could be formulated in terms of goals for which the actual Web services are discovered at runtime. Moreover, the tasks are very similar because most of the client applications work in the telecommunication domain. Thus, the relevant goal templates can be organized in a subsumption hierarchy so that the SDC technique can reveal its potential.

Another central aspect is the creation of goal templates as the central element of the SDC graph. While an initial goal template of a problem domain must be defined manually, it possible to generate further ones on basis of the used domain ontologies. In our running example, the ontologies define several continents, countries, and weight classes. Given a goal template for package shipment in Europe, we can generate goal templates for shipment in each European country, for other continents, and so on. Eventually, we can generate all goal templates that can be expressed by the domain ontologies. These naturally are semantically similar, and thus will constitute a very fine-grained subsumption hierarchy in the resulting SDC graph. This enhances the achievable performance increase for Web service discovery: the more semantically similar goal templates are in the SDC graph, the higher is its effectiveness for reducing the search space and minimizing the number of necessary matchmaking operations.

5 Related Work

Although there is a wealth of work on Semantic Web services and semantically enabled Web service discovery, we are not aware of any work that addresses the performance challenge for discovery in a similar way. The following outlines the foundations of our approach and positions it within related works.

The concept of goals as an abstraction layer for facilitating problem-oriented client-system interaction has initially been developed in AI technologies like BDI agents and cognitive architectures. Inspired by the works on UPML [6], our approach has been developed in the spirit of the WSMO framework that promotes a goal-driven approach for Semantic Web services [5], and the IRS system that provides a goal-based broker for Web service usage [2]. We have integrated these approaches, and extended them with a sufficiently rich formalization and the caching mechanism for Web service discovery.

Discovery and Semantic Matchmaking. This has been subject to many research works that provide valuable insights on several aspects, e.g. on the architectural allocation in SWS systems [15], the quality criteria of discovery [13,9], and semantic matchmaking for different logical languages [11,14,10]. Our contribution towards this end is the two-phase Web service discovery with precise formal semantics and adequate matchmaking techniques ([17], *cf.* Section 2).

Web Service Clustering. Other, not goal-based approaches aim at reducing 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: it can not be ensured that the classification scheme properly

reflects the functionalities provided by Web services. More sophisticated solutions perform clustering on the basis of formal descriptions. E.g. [4] creates a search tree on based so-called interval constraints that describe Web services. These are significantly less expressive than our functional descriptions. Besides, although a logarithmic search time may be achieved (if the tree is balanced), still matchmaking is required for each new incoming request. The SDC technique can detect usable Web services without invoking a matchmaker.

Caching. Caching techniques are a well-established means for performance optimization applied in several areas of computing, among others also for increasing the efficiency of reasoning techniques (e.g. [1,3]). Respective studies show that caching can achieve the highest efficiency increase if there are many similar requests [7]. This complies with the design of our approach: the SDC graph provides the cache structure for Web service discovery, and the more similar goals and Web services exists, the higher is the achievable optimization.

6 Conclusions

This paper has presented a novel approach for enhancing the computational performance of Web service discovery by applying the concept of caching. We capture the minimal knowledge on the functional usability of available Web services for goal templates as generic, formal objective descriptions. At runtime, a concrete client request is formulated as a goal instance that instantiates a goal template with concrete inputs. The captured knowledge is used for optimizing the detection of usable Web services. The approach is based on a profound formal model for semantically enabled discovery. An evaluation with real-world data shows that our technique can help in the realization of scalable and reliable automated discovery engines, which becomes important for their employment as a heavily used component in larger, semantically enabled SOA systems. For the future, we plan to adopt the model to other specification languages and further integrate the caching mechanism into Semantic Web services environments.

Acknowledgements. The work presented in this paper has been supported by the European Commission under the projects SUPER (FP6-026850), SWING (FP6-26514), and MUSING (FP6-027097), and by the Austrian BMVIT/FFG under the FIT-IT project myOntology (Grant no. 812515/9284). The authors thank Holger Lausen, Stijn Heymans, and Dieter Fensel for fruitful discussions, and Michael Brodie (Verizon Chief Scientist) for the provision of information.

References

1. O. L. Astrachan and M. E. Stickel. Caching and Lemmaizing in Model Elimination Theorem Provers. In *Proc. of the 11th International Conference on Automated Deduction (CADE-11)*, 1992.
2. L. Cabral, J. Domingue, S. Galizia, A. Gugliotta, B. Norton, V. Tanasescu, and C. Pedrinaci. IRS-III – A Broker for Semantic Web Services based Applications. In *Proc. of the 5th International Semantic Web Conference (ISWC 2006), Athens(GA), USA*, 2006.

3. R. Clayton, J. G. Cleary, B. Pfahringer, and M. Utting. Tabling Structures for Bottom-Up Logic Programming. In *Proc. of 12th International Workshop on Logic Based Program Synthesis and Tranformation, Madrid, Spain*, 2002.
4. 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.
5. D. Fensel, H. Lausen, A. Polleres, J. de Bruijn, M. Stollberg, D. Roman, and J. Domigue. *Enabling Semantic Web Services. The Web Service Modeling Ontology.* Springer, Berlin, Heidelberg, 2006.
6. D. Fensel et al. The Unified Problem Solving Method Development Language UPML. *Knowledge and Information Systems Journal (KAIS)*, 5(1), 2003.
7. P. Godfrey and J. Gryz. Semantic Query Caching for Heterogeneous Databases. In *Proc. of 4th Knowledge Representation Meets Databases Workshop (KRDB) at VLDB'97, Athens, Greece*, 1997.
8. M. Hepp, F. Leymann, J. Domingue, A. Wahler, and D. Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *Proc. of the IEEE ICEBE 2005, Beijing, China*, 2005.
9. U. Keller, R. Lara, H. Lausen, and D. Fensel. Semantic Web Service Discovery in the WSMO Framework. In J. Cardoses, editor, *Semantic Web: Theory, Tools and Applications*. Idea Publishing Group, 2006.
10. M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *Proc. of the ISWC 2004 workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, Hiroshima, Japan*, 2004.
11. L. Li and I. Horrocks. A Software Fframework for Matchmaking based on Semantic Web Technology. In *Proceedings of the 12th International Conference on the World Wide Web, Budapest, Hungary*, 2003.
12. H. Lu. Semantic Web Services Discovery and Ranking. In *Proc. of the ACM International Conference on Web Intelligence(WI 2005)*, Compiegne, France, 2005.
13. T. Di Noia, E. Di Sciascio, F. Donini, and M. Mongiello. A System for Principled Matchmaking in an Electronic Marketplace. In *Proc. of the 12th International Conference on the World Wide Web (WWW'03), Budapest, Hungary*, 2003.
14. M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic Matching of Web Services Capabilities. In *Proc. of the 1st International Semantic Web Conference, Sardinia, Italy*, 2002.
15. C. Preist. A Conceptual Architecture for Semantic Web Services. In *Proc. of the 2nd International Semantic Web Conference (ISWC 2004)*, 2004.
16. M. Stollberg and Martin Hepp. Semantic Discovery Caching: Prototype & Use Case Evaluation. Technical Report DERI-2007-03-27, DERI, 2007.
17. M. Stollberg, U. Keller, H. Lausen, and S. Heymans. Two-phase Web Service Discovery based on Rich Functional Descriptions. In *Proc. 4th European Semantic Web Conference (ESWC 2007), Innsbruck, Austria*, 2007.
18. M. Stollberg and B. Norton. 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.
19. P. Traverso and M. Pistore. Automatic Composition of Semantic Web Services into Executable Processes. In *Proc. 3rd International Semantic Web Conference (ISWC 2004), Hiroshima, Japan*, 2004.
20. L.-H. Vu, M. Hauswirth, and K. Aberer. QoS-Based Service Selection and Ranking with Trust and Reputation Management. In *Proc. of the OTM Confederated International Conferences CoopIS, DOA, and ODBASE 2005, Cyprus*, 2005.