Volume 1, No. 3, Sept-Oct 2010
ISSN No. 0976-5697
International Journal of Advanced Research in Computer Science
RESEARCH PAPER
Available Online at www.ijarcs.info

A novel method for semantic web services discovery

Kambiz Fakhr*
Maragheh Islamic Azad University
Tabriz, Iran
kambiz@fakhr.org

Dr.Saeid Parsa
Iran University of Science & Technology,
Tehran, Iran,
parsa@iust.ac.ir

Abstract: The discovery of suitable web services for a given task is one of the major operations in SOA architecture, and researches are being done to automate 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 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 exploit 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.

Keywords: Web services discovery; Semantic web services; Caching mechanism; SDC graph; BDI agents

I. INTRODUCTION

A Web service is a software system identified by a URI whose public interface and bindings are defined and described by XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by Internet protocols.

The problem of composing Web services can be reduced to three fundamental problems: the first one is to make a *plan* that describes how Web services interact and how the functionality they offer can be integrated to provide a solution of the problem. The second problem is the *discovery* of the Web services that perform the tasks required in the plan. The third problem is the management of the *interaction* with those Web services. Crucially, planning, discovery and interaction are strictly interconnected: a plan specifies the type of Web services to discover, but it also depends on the Web services that are available. Similarly, the interaction process depends on the specifics of the plan, but the plan itself depends on the requirements of the interaction.

Web service discovery defined as finding a suitable web services for a given task, and it is one of the major operations in service oriented architecture (SOA). Nowadays, important works are done in the field semantic web service (SWS) for discovering automatic web services that are mainly focused on important and efficient retrieval of web services (e.g. [14, 11, 13, 9]). However, the improvement and computational performance at SWS are neglected. According to increasing amount of available web services in real world, and using of semantic discovery engine for dynamic web service composition and business process management, discovery engine would be change to a bottleneck. So by increasing amount of web services, request for discovery engine will be increase.

The following characteristics for judging the computational reliability of a discovery engine are taken into account: 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 the stability as a low variance of the execution time of several invocation of same service.

In this paper, we presented a method for dealing with above mentioned cases and discovering of web services by the use of caching and reasoning concepts. 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.

The given method is called semantic Discovery Caching (SDC). At the runtime, BDI reasoning engine activates the agents and these agents classify the services according to their quality. We used factors such as secure delivery of messages, response time and security for distinction between services at runtime. At the last step of discovery, we use the dynamic discovery method mentioned in [21] for matchmaking in runtime. Figure 1 shows the basic idea of this method by means of the data flow graph. There are three major entities in this figure as follow: web services, having formal description, goal templates that describe the general goals that are stored in the system, and goal instances that describe a real request by instantiating a goal template with real inputs. Web services are discovered for goal patterns in design time. The result is stored in a specific knowledge structure, called SDC graph. At run time, a client request formulated as a goal instance for witch suitable web services need to be discovered. Due to the most usage of discovery engine component by SOA applications, we tend to optimize the discovery engine by using of SDC graph in order to reduce the search space and minimize the number of necessary matchmaking operations.

The rest of the paper is organized as follows: section 2 briefly describes the approaches for semantically discovering web services ([17, 18]). Section 3 specifies the caching techniques of semantic discovery. Section 4 describe reasoning engine based on BDI agents, dynamic web service discovery and composition them using of SDC graph for making integrated discovery of the web services. Section 5 presents the evaluation and discuses the relevance and specifics of our approaches. At last, 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://sws-challenge.org/wiki/index.php/Main_Page).

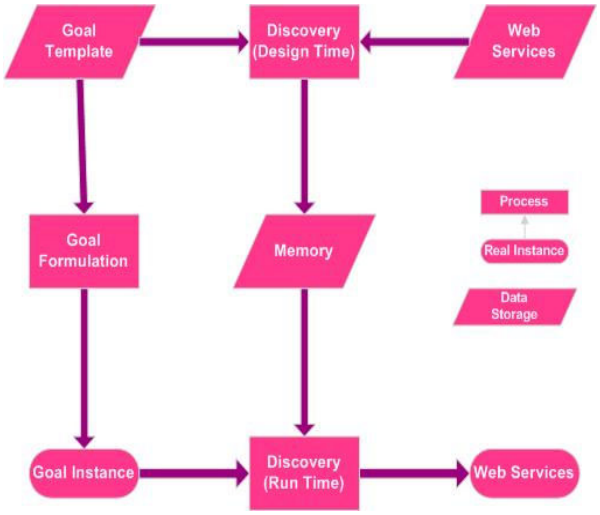

Figure 1 .DFD for semantic web services discovery

II. Basis of the Discovery Framework

This section presents the next part of our approach and is regarded as a basis for it. We have developed an intended method 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 easing 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 provide precise definitions of goals, Web services, and the necessary matchmaking techniques. We here recall the central aspects; the full model is defined in [17].

A. 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 $\tau = (S_0,....,S_m)$, 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 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 D over a signature $\sum$, and use ontology Ω as the background knowledge. D consists of a set of input variables $IN = \{i_1,....,i_n\}$, 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 D is defined as implication semantics between the precondition and the effect. We say that a Web service W provides the functionality described by D, denoted by $W \models D$, if and only if for all $\tau \in \{\tau\}_W$ holds that if $S_0 \models \phi^{Pre}$ then $S_m \models \phi^{eff}$. In order to deal with functional descriptions in terms of model-theoretic semantics, we present this as a FOL formula ϕ^D of the form $\phi^{pre} \Rightarrow \phi^{eff}$. Then, $W \models D$ is given if and only if every $\tau \in \{\tau\}_W$ is represented by a $\sum$ -interpretation that is a model of ϕ^D. Analogously, the functional description DG 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 instances 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 $\beta : \{i_1...i_n\} \rightarrow u$ as a total function that assigns objects of U to each IN -variable of DG. 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 ϕ^{D_G} is satisfying 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* $GI \models G \rightarrow \{\tau\}_{GI(G)} \subseteq \{\tau\}_G$.

Goal Template G "ship a package of any weight in Asia"	**Web Service W "shipment in Iran , max 50 kg"**
Ω : location & shipment ontolog IN : {?s, ?r, ?p, ?w} ϕ^{pre} : person(?s) ^ in(?s, asi ^ person(?r) ^in(?r;Asia) ^ package(?p) ^ weight(?p; ?w)^ maxWeight(?w; heavy): ϕ^{eff} : ∀ ?o, ?price : out(?o) ⇐ shipmentOrder(?o; ?p) ^ sender(?p; ?s) ^ receiver(?p; ?r) ^ costs(?o; ?price)).	Ω : location & shipment ontology IN : {?s, ?r, ?p, ?w} ϕ^{pre} : person(?s) ^ in(?s, Iran) ^ person(?r) ^ in(?r; Iran) ^ package(?p) ^ weight(?p; ?w) ^ maxWeight(?w; heavy): ϕ^{eff} : ∀ ?o, ?price : out(?o) ⇔ (shipmentOrder(?o; ?p) ^ sender(?p; ?s) ^ receiver(?p; ?r) ^ costs(?o; ?price)).

Table 1 functional descriptions

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].

[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 *D* to be *consistent*, i.e. that ϕ^D is satisfying under an input binding β. Otherwise, a Web service $W \models D$

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* $\exists \tau \mid \tau \in \{\tau\}_G \bigcap \{\tau\}_W$. We express the usability of W for G in terms of matching degrees as defined in Table 2. Four degrees {exact, plug-in, 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

(4) $\neg$ Intersect $\Leftrightarrow$ Disjoint

Table 2

Denotation	Definition	Meaning
Exact(Dg ,DW)	$\Omega \models \forall \beta . \phi^{D_g} \Leftrightarrow \phi^{Dw}$	$\tau \in \{\tau\}_G \Leftrightarrow \tau \in \{\tau\}_w$
Plugin (Dg ,DW)	$\Omega \models \forall \beta . \phi^{D_g} \Rightarrow \phi^{Dw}$	$\tau \in \{\tau\}_G \Rightarrow \tau \in \{\tau\}_w$
Subsume(Dg ,DW)	$\Omega \models \forall \beta . \phi^{D_g} \Leftarrow \phi^{Dw}$	$\tau \in \{\tau\}_G \Leftarrow \tau \in \{\tau\}_w$
Intersect (Dg ,DW)	$\Omega \models \exists \beta . \phi^{D_g} \wedge \phi^{Dw}$	there is a τ such that $\tau \in \{\tau\}_G AND \tau \in \{\tau\}$
Disjoint(Dg ,DW)	$\Omega \models \neg \exists \beta . \phi^{D_g} \wedge \phi^{Dw}$	there is no τ such that $\tau \in \{\tau\}_G AND \tau \in \{\tau\}$

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 $\Omega \cup \{[\phi^{D_G}]_\beta, [\phi^{D_W}]_\beta\}$ is satisfying. This means that under consideration of the domain knowledge Ω and under the input binding β defined in GI(G) there must be a $\sum$ -interpretation that represents a solution for the corresponding goal template G as well as a possible execution of the Web service W. However, we 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, β) be a goal instance with GI(G) $\models$ G. Let W be a Web service, and let DW be a functional description such that W $\models$ DW.

W is usable for solving GI(G) if and only if:

(i) Exact (DG, DW) or

(ii) Plug-in (DG, DW) or

(iii) Subsume (DG, DW) and $\Omega \wedge [\phi^{D_W}]_\beta$ is satisfying, or

(iv) Intersect (DG, DW) and $\Omega \wedge [\varphi^{D_G}]_\beta \wedge [\varphi^{D_W}]_\beta$ is satisfying.

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 plug-in 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.

III. 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 re-liability 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.

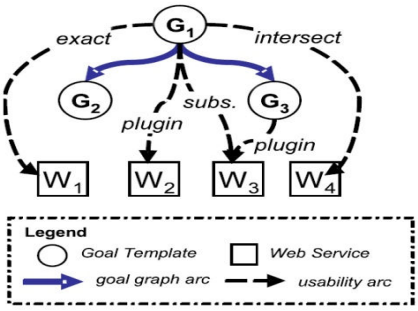

Figure 2.Example for cache graph

A. 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 Gi and Gj 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(Gi;Gj) that denote the matching degree between the functional descriptions D_{Gi} and D_{Gj}. 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 (Gi,Gj). If this is given, then (1) the solutions for the child Gj are a subset of those for the parent Gi, and thus (2) the Web services that are usable for Gj are a subset of those usable for Gi.

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: G1 for package shipment in Asia, G2 for China, and G3 for Iran. Their similarity degrees are subsume (G1, G2) and subsume (G1, G3), which is explicated in the goal graph. Consider some Web services, e.g. W1

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.

for package shipment in Asia, W2 in the whole world, W3 in the Middle east, and W4 in the Common wealth. 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.

B. 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 (Gi, Gj) denote the similarity degree of goal templates Gi and Gj, and let d (G, W) denote the usability degree of a Web service W for a goal template G. Given a set G of goal templates and a set W of Web services, the SDC graph is a directed acyclic graph $(V_g \cup V_W, E_{sim} \cup E_{use})$ such that:

(i) $V_g := g \cup g^I$ is the set of inner vertices where:

- $g = \{G_1, ..., Gn\}$ are the goal templates; and
- $g^I = \{G^I \mid G_i, G_j \in g, d(G_i, G_j) = \text{Intersect}, G^I = G_i \cap G_j\}$ is the set of intersected goal templates from G

(ii) $V_W = \{W_1, ..., W_m\}$ is the set of leaf vertices representing Web services

(iii) $E_{sim} = \{(G_i, G_j) \mid G_i, G_j \in V_g\}$ is the set of directed arcs where:- d(Gi;Gj) = subsume; and

- Not exists $d(G, G_j) = \text{Subsume}, \quad d(G_i, G) = \text{Subsume}$.

(iv) $E_{use} := \{(G, W) \mid G \in V_g, W \in V_W\}$ is set of directed arcs where:

- $d(G, W) \in$ {exact; plug-in; subsume; intersect}; and
- Not exists $G_i \in V_g$ s.t. d (Gi, G) = subsume, $d(G_i, W) \in \{\text{exact, plugin}\}$.

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 (Gi, Gj), cf. clause (iii). This denotes that Gj is a functional specialization of Gi such that $\{\tau\}_{G_j} \subset \{\tau\}_{G_i}$. In consequence, only those Web services that are usable for Gi can be usable for Gj because if $\{\tau\}_{G_j} \cap \{\tau\}_W = \phi$ then also $\{\tau\}_{G_i} \cap \{\tau\}_W = \phi$. 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 (Gi, Gj), only one goal template is kept while the other one is redundant; if plug-in(Gi , Gj) then we store the opponent arc (Gj , Gi). If disjoint (Gi, Gj), then both are kept as disconnected nodes in the SDC graph. Effectively, each of its connected sub graphs covers a problem domain, e.g. one for the shipment scenario and another one for flight ticketing.

The only critical similarity degree is intersect (Gi, Gj); denoting that Gi and Gj 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 GI (Gi, Gj) whose solutions are exactly those that are common to Gi and Gj , cf. clause (i). Formally, GI is defined as the conjunction of the functional descriptions of the original goal templates, i.e.

$$\varphi^{D_{G^I(G_i,G_j)}} = \varphi^{D_{G_i}} \wedge \varphi^{D_{G_j}} \longrightarrow \{\tau\}_{G^I(G_i,G_j)} = \{\tau\}_{G_i} \cap \{\tau\}_{G_j}$$

Because of this, it holds that

Subsume $(G_i, G^I(G_i, G_j))$, Subsume $(G_j, G^I(G_i, G_j))$ Thus, GI becomes a child node of both Gi and Gj in the goal graph. This is applied for every occurring intersects 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 minimalist 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 (Gi, Gj), then the usability degree of a Web service W for the child Gj is always plug-in if W is usable for the parent Gi under the degrees exact or plug-in because $\{\tau\}_W \supseteq \{\tau\}_{G_i} \supset \{\tau\}_{G_j}$. Thus, the arc (Gj, W) is not explicated in the SDC graph. In the above example, the Web services W1 and W2 are usable under the plug-in degree for both G2 and G3; this can be inferred from the usability cache arcs of G1. 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 modifications of goal templates are manual maintenance operations; respective algorithms are ensuring 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]

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 ;
```

List 1

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

[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.

that this is a consistent instantiation of its corresponding goal template G; if this is not given, a usable Web service cannot 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 plug-in 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 (G1) be a goal instance for shipping a package from Shiraz to Tehran that instantiates G1 for package shipment within Asia. This is also a proper instantiation of G3 for shipment within Germany; hence, we refine the goal instance to GI(G3). 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)|= G'. If there is an intersection goal template GI and GI(G) |= GI , 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). If there are no web services, the Smart Agent will be activated.

IV. SMART AGENTS

We use three following agents that their each operation will be described:

- Management Agent
- Sender Agent
- Ranking agent

Management agent is responsible for managing the message sending and ranking the services and this agent also assumes the responsibility for control issues. The control issues are as follow:

Activating the sender and ranking agents

Discharging the sender agent from the cycle of detection process during the running time and its correspondent ranking agent in case of Timeout of sending and receiving a message

Selecting an appropriate service with high ranking among those services which are received ranking agents.

The sender agent is led to a service request from providers and this message will be broadcasted to all service providers via SOAP protocol with XML format. Then, each provider receives the message which is sent by our system through Listener section and it returns the relevant WSDL. Ranking agents are responsible for receiving and ranking the services WSDLs and these agents rank their services according to quality.

The ranking agents use three components of service quality as follow:

Services security

Responding time

Secure sending of messages

The advantage of this conformity is that we can find the services using agents which are made when the SDC graph had been created. In fact, this section will be placed between the SDC graph and Prover which are already used before forming the target graph and If SDC couldn't find the requested service, so agents will be activated in order to detect the considered service during the running time. However, even if the service wouldn't be detected in this stage, the considered service will be no longer available. We selected the agents according to BDI agents and Jadax (6) is used as a class in eclips in order to change and use them, thus it is easily applied along with WSMO which is used in eclips.

V. EVALUATIONS

In order to evaluate the performance gain achievable with the SDC technique, we have compared the 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.

A. 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 plug-in 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.

B. 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 and 4 shows a fragment of the augmented data of all test runs for all ten goal instances.

Table 3 Our Approach

Covariance	Standard Division	Variance	Max	Min	Median	Average	Service no
11.74%	0.032998	1562.565	0.4563	0.2559	0.2734	0.281165	10
33.19%	0.097271	48644.89	0.9346	0.2574	0.27885	0.29307	20
11.79%	0.03356	1568.855	0.4595	0.2575	0.27885	0.284648	50
11.53%	0.033145	1489.189	0.4533	0.2651	0.28125	0.287544	100
11.51%	0.03339	1759.133	0.4359	0.2636	0.2815	0.290118	200
13.42%	0.039231	2384.443	0.4922	0.2652	0.2859	0.29223	500
14.79%	0.043867	2795.679	0.539	0.2651	0.2875	0.296686	1000
17.34%	0.052347	4701.46	0.594	0.2717	0.2906	0.301862	1500
18.03%	0.055162	4376.717	0.5954	0.2731	0.2938	0.306008	2000

Table 4 Non-Optimized

Covariance	Standard Division	Variance	Max	Min	Median	Average	Service no
51.71%	0.213582	49210.51	0.9438	0.1544	0.3922	0.413052	10
63.45%	0.51723	292723.9	2.3708	0.159	0.733	0.815156	20
64.59%	1.29219	1759017	5.054	0.1888	1.79795	2.000464	50
64.48%	2.554986	7027879	10.4883	0.253	3.67855	3.962612	100
66.35%	5.047952	26774771	19.5459	0.3484	6.6697	7.60848	200
72.34%	13.26301	1.87E+08	51.321	0.7014	15.61145	18.3343	500
69.70%	26.27577	7.26E+08	103.7427	2.1126	33.22145	37.69667	1000
72.84%	39.26856	1.63E+09	148.7762	1.4599	45.19965	53.90973	1500
71.45%	52.13448	2.9E+09	192.9702	2.3656	65.55615	72.96355	2000

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.

VI. Related Works

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.

A. 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).

B. 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 cannot 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.

C. 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.

VII. 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 instances 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 evalua-

tion 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.

VIII. 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] 4Constantinescu, W. Binder, and B. Faltings. Flexible and Enceinte 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 Framework 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.

[21] Co-Developers. Web Services Dynamic Discovery (ws-Discovery) . Microsoft Corporation, Inc. October 2004 .