# Efficient Discovery of Services
# Specified in Description Logics Languages

Claudia d'Amato[1], Steffen Staab[2], Nicola Fanizzi[1], and Floriana Esposito[1]

[1] Department of Computer Science, University of Bari, Italy
{claudia.damato, fanizzi, esposito}@di.uniba.it
[2] ISWeb, University of Koblenz-Landau, Germany
staab@uni-koblenz.de

**Abstract.** Semantic service descriptions are frequently given using expressive ontology languages based on description languages. The expressiveness of these languages, however, often implies problems for efficient service discovery, especially when increasing numbers of services become available in large organizations and on the Web. To remedy this problem, we propose an efficient service discovery/retrieval method grounded on a conceptual clustering approach, where services are specified in Description Logics as class definitions [10] and they are retrieved by defining a class expression as a query and by computing the individual subsumption relationship between the query and the available descriptions. We present a new conceptual clustering method that constructs tree indices for clustered services, where available descriptions are the leaf nodes, while inner nodes are intensional descriptions (generalization) of their children nodes. The matchmaking is performed by following the tree branches whose nodes might satisfy the query. The query answering time may strongly improve, since the number of retrieval steps may decrease from $O(n)$ to $O(log\ n)$ for concise queries. We also show that the proposed method is sound and complete.

## 1 Motivation

First research efforts in the fields of service discovery and service matchmaking have been concentrated on setting up methods and formalisms for describing the service semantics, with the goal of making service retrieval an automatic or, at least semi-automatic, task. Expressive ontology languages such as OWL-DL[3] are increasingly used to describe the ontology of a domain as well as specific resources available in such a domain such as Web services [16, 10]. Thereby such resources may be described as parts of an ABox, i.e. as instances of concept expressions [16]. Frequently, however, it is also useful to fully exploit Description Logics (DLs) subsumption capabilities using concept expressions in a TBox, e.g. to describe all the possible concretizations of a specific Web service resource (as proposed in [10]) using TBox concept expressions to represent the generic Web service resources (and their instance to represent Web service instantiations). Eventually, the services are retrieved by formulating a request as a concept expression and checking each service description with regard to instantiation or

---

[3] www.w3.org/2004/OWL/

subsumption of the query. However, these approaches suffer heavily from runtime inefficiency when the number of available services grows, since the number of instantiation or subsumption checks grows linearly in the number of the available resources.

In spite of such limitations, most of the current researches that concentrate on the automation of the service discovery and retrieval focus on the improvement of the effectiveness of the matchmaking process rather than on the efficiency of the whole retrieval process. Specifically, central to the majority of the current service matchmaking approaches is that the formal semantics of service descriptions is explicitly defined in an ontology language such as OWL-DL [12] based on some decidable DLs [1]. In this way service matchmaking can be performed by exploiting standard DL inferences [16, 22, 7, 13] sometimes jointly with the use of syntactic similarity measures [14].

Less attention has been dedicated to the improvement of the efficiency of the service discovery task. In this paper we address such a problem, by proposing an *original tree indexing method, **DL-tree**, for services described as DLs concept expressions* and referring to an ontology acting as common knowledge base. The DL-tree is obtained by exploiting *a novel conceptual clustering algorithm for concept expressions*, **DL-link**, grounded on the use of *a new, truly semantic similarity measure*, **GCS-based similarity**.

In the DL-tree, available service descriptions are found at the leaf nodes, while inner nodes are generalizations of their children nodes. Germane to the heuristic construction of DL-tree are non-standard inference procedures for computing the Good Common Subsumer [3] of $\mathcal{ALE}(\mathcal{T})$ concept expressions (see Sect. 2). Hence, in this paper, we focus on the description and evaluation of services specified in this ontology language[4]. Since the DL-tree is computed on the ground of the GCS-based similarity, it is different from the subsumption-based taxonomy representing the ontology.

Once that the DL-tree for a set of service description is obtained, its structure may be used to focus subsumption-based matching to the branches of the tree where nodes satisfy subsumption (or instantiation) checks. By this way, we achieve a drastic reduction of the size of the retrieval space. Query answering time may strongly improve, since the number of retrieval steps may decrease from $O(n)$ to $O(log\ n)$ for concise queries and $n$ resources. Because of the way that the DL-tree is constructed it heuristically achieves a good covering of the retrieval space while maintaining soundness and completeness of the retrieval method.

In the following, we first summarize some important definitions of the DLs representation languages, $\mathcal{ALC}$ and $\mathcal{ALE}$, and some established reasoning procedures that we use subsequently (Sect. 2). We define a semantic similarity measure, *GCS-based similarity*, for $\mathcal{ALE}(\mathcal{T})$ in Sect. 3. The measure combines extensional size of concept expressions to reflect their model semantics and an intensional generalization of two concepts, their *Good Common Subsumer* (cf. [3]) to consider also the structure of the given ontology. The GCS-based similarity is used to cluster service descriptions into the new indexing structure, DL-tree (instead of the DAG given by the ontology), using a modified agglomerative clustering mechanism in Sect. 4. The DL-tree is exploited in the service retrieval procedure defined in Sect. 5. In Sect. 6 a preliminary experi-

---

[4] In the conclusion we will indicate some ways to generalize DL-tree to indexing of OWL-DL concept expressions and instances.

mental evaluation of the DL-tree indexing method is presented. Some related works are examined in Sect. 7 while conclusions and future extensions of DL-Tree indexing are presented in Sect. 8.

## 2 The Reference Representation Language

We recall the basics of $\mathcal{ALC}$ and $\mathcal{ALE}$ [1] logics which adopt constructors supported by the standard ontology languages (see the DL handbook [1] for a thorough reference). In DLs, descriptions are inductively defined starting with a set $N_C$ of *primitive concept names* and a set $N_R$ of *primitive roles*. The semantics of the descriptions is defined by an *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set representing the *domain* of the interpretation, and $\cdot^{\mathcal{I}}$ is the *interpretation function* that maps each $A \in N_C$ to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and each $R \in N_R$ to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The *top* concept $\top$ is interpreted as the whole domain $\Delta^{\mathcal{I}}$, while the *bottom* concept $\bot$ corresponds to $\emptyset$. Complex descriptions can be built in $\mathcal{ALC}$ using primitive concepts and roles and the following constructors whose semantics is also specified. The language supports *full negation*, denoted $\neg C$ (given any description $C$), it amounts to $\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$. The *conjunction* of concepts, denoted $C_1 \sqcap C_2$, yields an extension $C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$ and, dually, concept *disjunction*, denoted $C_1 \sqcup C_2$, yields the union $C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$. Finally, the *existential restriction*, denoted $\exists R.C$, is interpreted as the set $\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}}((x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$ and the *value restriction* $\forall R.C$, has the extension $\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}}((x,y) \in R^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}})\}$.

$\mathcal{ALE}$ logic is a sub-language of $\mathcal{ALC}$ as only a subset of $\mathcal{ALC}$ constructors is allowed. Specifically, concept disjunction is not allowed and only the *atomic negation* can be used, namely complex concept descriptions cannot be negated.

The main inference in DLs is *subsumption* between concepts:

**Definition 1 (subsumption).** *Given two descriptions $C$ and $D$, $C$ subsumes $D$, denoted by $C \sqsupseteq D$, iff for every interpretation $\mathcal{I}$ it holds that $C^{\mathcal{I}} \supseteq D^{\mathcal{I}}$. When $C \sqsupseteq D$ and $D \sqsupseteq C$ then they are equivalent, denoted with $C \equiv D$.*

A *knowledge base* $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ contains a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$. $\mathcal{T}$ is the set of definitions $C \equiv D$, meaning $C^{\mathcal{I}} = D^{\mathcal{I}}$, where $C$ is the concept name and $D$ is its description. $\mathcal{A}$ contains assertions on the world state, e.g. $C(a)$ and $R(a,b)$, meaning that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. Subsumption based axioms ($D \sqsubseteq C$) are also allowed in the TBoxes as partial definitions. Indeed, $C \equiv D$ amounts to $D \sqsubseteq C$ and $C \sqsubseteq D$.

A related inference used in the following is *instance checking*, that is deciding whether an individual is an instance of a concept [1]. Concept subsumption and instance checking are examples of standard inference services in DLs. Besides of these, we also use non-standard inference services.

The most used non-standard inference services are the *Most Specific Concept* of an individual and the *Least Common Subsumer* of a given collection of concepts.

**Definition 2 (Most Specific Concept).** *Given an ABox $\mathcal{A}$ and an individual $a$, the most specific concept of $a$ w.r.t. $\mathcal{A}$ is the concept $C$, denoted $MSC_{\mathcal{A}}(a)$, such that $\mathcal{A} \models C(a)$ and $\forall D$ such that $\mathcal{A} \models D(a)$, it holds: $C \sqsubseteq D$.*

In the general case of a cyclic ABox expressed in an expressive DL endowed with existential or numeric restriction, the MSC cannot be expressed as a finite concept description [1], thus it can only be approximated. Generally an approximation of the MSC is considered up to a certain depth $k$. The maximum depth $k$ has been shown to correspond to the depth of the considered ABox, as defined in [17]. We will indicate generically an approximation to the maximum depth with $MSC^*$.

**Definition 3 (Least Common Subsumer).** *Let $\mathcal{L}$ be a description logic. A concept description $E$ of $\mathcal{L}$ is the **least common subsumer (LCS)** of the concept descriptions $C_1, \cdots, C_n$ in $\mathcal{L}$ ($LCS(C_1, \cdots, C_n)$ for short) iff it satisfies:*

1. *$C_i \sqsubseteq E$ for all $i = 1, \cdots, n$ and*
2. *$E$ is the least $\mathcal{L}$-concept description satisfying (1), i.e. if $E'$ is an $\mathcal{L}$-concept description satisfying $C_i \sqsubseteq E'$ for all $i = 1, \cdots, n$, then $E \sqsubseteq E'$.*

Depending on the DL language, the LCS need not always exist. If it exists, it is unique up to equivalence. In the case of $\mathcal{ALC}$ and $\mathcal{ALE}$ logic, the LCS always exists [2, 1]. Specifically, in the case of $\mathcal{ALC}$ (as in every DL allowing for concept disjunction) the LCS is given by the disjunction of the considered concepts, thus it is also "uninteresting" as it does not reflect any ontological modeling decisions. In the case of $\mathcal{ALE}$ logic, where disjunction is disallowed, the LCS is computed by taking the common concept names in the concept descriptions (also in the concepts scope of universal and existential restrictions w.r.t. the same role), without considering the TBox (see [2] for more details).

The $\mathcal{ALE}$ LCS computed using such an approach often results to be very general. For this reason the notion of LCS computed w.r.t. the TBox[5] to which the concept definitions refer has been introduced [3].

**Definition 4 (LCS w.r.t. a TBox).** *Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be DLs such that $\mathcal{L}_1$ is a sub-DL of $\mathcal{L}_2$. i.e., $\mathcal{L}_1$ allows for less constructors. For a given $\mathcal{L}_2$-TBox $\mathcal{T}$, let $\mathcal{L}_1(T)$-concept descriptions be those $\mathcal{L}_1$-concept descriptions that may contain concepts defined in $\mathcal{T}$. Given an $\mathcal{L}_2$-TBox $\mathcal{T}$ and $\mathcal{L}_1(T)$-concept descriptions $C_1, \ldots, C_n$, the least common subsumer (LCS) of $C_1, \ldots, C_n$ in $\mathcal{L}_1(T)$ w.r.t. $\mathcal{T}$ is the most specific $\mathcal{L}_1(T)$-concept description that subsumes $C_1, \ldots, C_n$ w.r.t. $\mathcal{T}$, i.e., it is an $\mathcal{L}_1(T)$-concept description $D$ such that:*

1. *$C_i \sqsubseteq_\mathcal{T} D$ for $i = 1, \ldots, n$*
2. *if $E$ is an $\mathcal{L}_1(T)$-concept description satisfying $C_i \sqsubseteq_\mathcal{T} E$ for $i = 1, \ldots, n$, then $D \sqsubseteq_\mathcal{T} E$*

Specifically, in [3], the case of $\mathcal{L}_2 = \mathcal{ALC}$ and $\mathcal{L}_1 = \mathcal{ALE}$ has been focused and it has been proved that the $\mathcal{ALE}$ LCS w.r.t. an *acyclic $\mathcal{ALC}$ TBox*[6] always exists, while it cannot exist in case of *cyclic* or *general*[7] TBoxes. A brute force algorithm for computing the LCS w.r.t. a TBox has been also shown. Anyway, such an algorithm cannot be usable

---

[5] The TBox can be described by a DL that is more expressive than $\mathcal{ALE}$.

[6] A TBox is called acyclic if it does not contain any concept definition in which the concept name to define is also used in the concept definition. If this happens, the TBox is called cyclic.

[7] A TBox in which inclusion axioms are defined is called general TBox.

in practice. For this reason, an algorithm for computing an approximation of the $\mathcal{ALE}$ LCS w.r.t. an $\mathcal{ALC}$ TBox has been presented. The computed approximation is called *Good Common Subsumer* (GCS) w.r.t. a TBox and it can exist also when a general TBox is considered. The GCS is computed by determining the smallest conjunction of concept and their negations that is able to subsume the conjunction of the top level concept names of each considered concept, and the same for the concepts that constitute the range of existential and universal restrictions w.r.t. the same role. The GCS is more specific than the LCS computed by ignoring the TBox, though in general it subsumes (or is equivalent to) the least common subsumer w.r.t. the TBox (see [3] for more details).

## 3   GCS-based Similarity: A Semantic Similarity Measure for $\mathcal{ALE}(\mathcal{T})$ Concept Descriptions

In the last few years, several measures for assessing the similarity value between concepts have been proposed in the literature. From them, two main approaches can be distinguished: 1) measures based on semantic relations (also called path distance measures); 2) measures based on concept extensions and *Information Content*.

In the former approach all concepts are in an is-a taxonomy, and the similarity value between two concepts is computed by counting the (weighted) edges in the paths from the considered concepts to their most specific ancestor. Concepts with a few links separating them are similar; concepts with many links between them are less similar [23, 15, 6, 18].In the latter approach the similarity value is computed by counting the common instances of the concept extensions [8] or by measuring the variation of the *Information Content* between the considered concepts [9, 24, 5].

Since the ontology does not have the simple structure of a taxonomy, but it is rather an elaborated DAG, similarity measures based on distances in the taxonomy cannot be used. Furthermore, in the considered application domain, the typical scenario consists of having a set of concept descriptions (the service descriptions) with no one necessarily overlapping the others. Consequently also measures based on overlap of concept extensions as well as measures based on *Information Content* cannot be used, since also semantically similar concept could result to be totally different.

Hence, we define a new semantic similarity measure, the **GCS-based similarity**, able to cope with the presented scenario. Moving from the principles of the measures based on concept extension and Information Content, we propose a similarity measure exploiting the notion of concept extension, but instead of counting the common instances between two considered concepts, the similarity value is given by the variation of the number of instances in the concept extensions w.r.t. the number of instances in the extension of their common super-concept. The common super-concept is given by the GCS of the concepts (see Sect. 2). The measure is formally defined in the following.

**Definition 5 (GCS-based Similarity Measure).** *Let $\mathcal{T}$ be an $\mathcal{ALC}$ TBox. For all $C$ and $D$ $\mathcal{ALE}(\mathcal{T})$-concept descriptions, the* Semantic Similarity Measure $s$ *is a function* $s : \mathcal{ALE}(\mathcal{T}) \times \mathcal{ALE}(\mathcal{T}) \to [0, 1]$ *defined as follow:*

$$s(C, D) = \frac{\min(|C^I|, |D^I|)}{|(GCS(C,D))^I|} \cdot (1 - \frac{|(GCS(C,D))^I|}{|\Delta^I|} \cdot (1 - \frac{\min(|C^I|, |D^I|)}{|(GCS(C,D))^I|}))$$
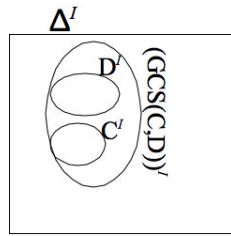
**Fig. 1.** Concepts $C \equiv$ credit-card-payment, $D \equiv$ debit-card-payment are similar as the extension of their GCS$\equiv$card-payment does not include many other instances besides of those of their extensions.
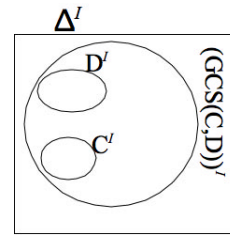
**Fig. 2.** Concepts $C \equiv$ car-transfer, $D \equiv$ debit-card-payment are different as the extension of their GCS$\equiv$service includes many other instances besides of those of the extension of $C$ and $D$.

*where $(\cdot)^I$ computes the concept extension w.r.t. the interpretation $I$ (canonical interpretation).*

The canonical interpretation adopts the set of individuals mentioned in the ABox as its domain and the identity as its interpretation function [1, 17].

The rationale of the presented measure is that if two concepts are semantically similar, such as credit-card-payment and debit-card-payment, then they should have a good common superconcept, e.g. card-payment, that is so close to the two concepts, namely the extensions of the superconcept and even the lesser-sized input concept share many instances, consequently the similarity value will be close to $1$. On the contrary, if the considered concepts are very different, e.g. car-transfer and debit-card-payment, their GCS, e.g. service, will be high up in the TBox, and this superconcept will have many instances not contained in the two compared concepts, consequently the similarity value will be next to $0$. In Fig. 1 and Fig. 2 this rationale is illustrated. Opposite to existing semantic similarity measures, this rationale does not require overlap of compared concepts, and it does not take into account semantic path distance. Moreover, the minimum extension of the concepts is considered in the measure definition, in order to avoid to have an incorrect similarity value (high similarity value) in the case in which one of the concepts is very similar to the super-concept but very different from the considered one.

It is trivial to verify that the function $s$ of Def. 5 is really a *similarity measure*, namely that (following the formal definition in [4]) it satisfies the following properties: 1) $s$ is definite positive; 2) $s$ is symmetric; 3) $s$ satisfies the maximality property $(\forall\, C, D\ :\ s(C, D) \leq S(C, C))$.

## 4 *DL-Link*: A Conceptual Hierarchical Agglomerative Algorithm for Clustering Description Logic Resources

The main goal of Cluster Analysis is to set up methods for the organization of a collection of unlabeled resources into meaningful clusters exploiting a similarity criterion. Specifically, clusters (classes) are collections of resources whose intra-cluster similarity

is high and inter-cluster similarity is low. The methods that focus also on techniques for generating intensional cluster description are called *Conceptual Clustering methods*.

A prominent conceptual clustering algorithm is the *average-link* algorithm [26]. It starts by assigning each resource to a distinct cluster and iteratively merges the two most similar clusters until only one cluster remains. The output of the algorithm is a dendrogram, namely a tree structure representing a nested grouping of resources. We modify average-link in several ways, in particular we substitute the typical Euclidean distance measure by the GCS-based similarity and we substitute the computation of the Euclidean average of each cluster in the computation of the GCS of the merged clusters.

Resources are assumed to be described as $\mathcal{ALE}(\mathcal{T})$ concepts, exploiting a common vocabulary $\mathcal{T}$ (mainly a shared ontology) written in $\mathcal{ALC}$ logic. They are clustered by a batch process by the use of the **DL-Link** algorithm detailed in the following.

Let $S = \{S_1, \ldots, S_n\}$ be a set of available resources.

1. Let $\mathcal{C} = \{C_1, \ldots, C_n\}$ be the set of initial clusters obtained by considering each resource as a single cluster
2. $n := |\mathcal{C}|$
3. For $i := 1$ to $n - 1$ consider cluster $C_i$
   - For $j := i + 1$ to $n$ consider cluster $C_j$
     - compute the similarity values $s_{ij}(C_i, C_j)$
4. compute $\max_{hk} = \max_{i,j=1,\ldots,n} s_{ij}$ where $h$ and $k$ are the clusters with maximum similarity
5. create the intensional cluster description $C_m = GCS(C_h, C_k)$
6. set $C_m$ as parent node of $C_h$ and $C_k$
7. insert $C_m$ in $\mathcal{C}$ and remove $C_h$ and $C_k$ from $\mathcal{C}$
8. if $|\mathcal{C}| \neq 1$ go to 2

DL-Link algorithm starts by considering each service description in a single cluster (list of available clusters), hence the similarity value[8] between all couples of clusters is computed and the couple having the highest similarity is selected. Then a new description, generalizing the chosen clusters, is created by computing the GCS (see Sect. 2). The new description is first set as parent node of the chosen clusters, then it is inserted in the list of the available clusters while the selected ones are removed. The generated description represents a cluster made by a single element while the resources that it describes are represented as its children in the dendrogram under construction. We call such a dendrogram **DL-Tree**. The same steps are iteratively repeated, until a unique cluster (describing all resources) is obtained.

An example of the DL-Tree returned by the DL-link algorithm is shown in Fig.3. It is a binary tree and this is because, at every step, only two clusters are merged[9]. Anyway, it can happen that two children nodes (or more than two) of the DL-Tree have the same intentional description as well as it can happen that a parent node has the same description as a child node. Since such redundant nodes do not add any information, a

---

[8] The GCS-based similarity measure presented in Sect. 3 is used.

[9] The clustering process could be speeded up by finding a way for merging more than two clusters at every step.
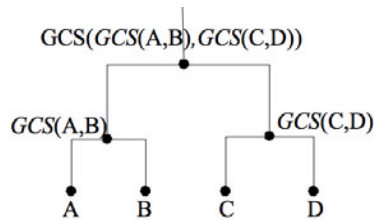
**Fig. 3.** DL-Tree index (dendrogram) returned by the hierarchical clustering algorithm presented in Section 4 applied to the four service descriptions respectively labeled by A, B, C, and D. *GCS* is the *Good Common Subsumer* of two $\mathcal{ALE}(\mathcal{T})$ descriptions.
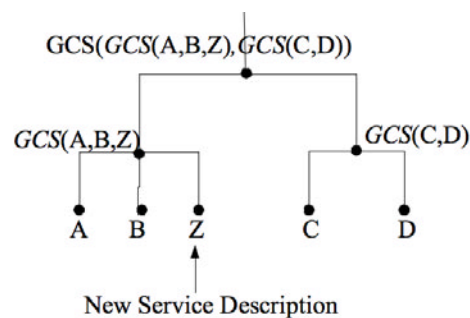


**Fig. 4. Z** is a new service description occurring after the DL-Tree construction. Once that the most similar service description **B** is found, **Z** is added has sibling node of **B** and the parent node of **B** is recomputed, as well as all parent node in the path, until the root node.

post-processing step is applied to the DL-Tree. If a child node is equal to another child node then one of them is deleted and their children nodes are assigned to the remaining node. If a child node is equal to a parent node then the child node is deleted and its children nodes are added as children of its parent node. The result of this flattening process is an n-ary DL-Tree.

It is important to note that, in case of a new service description occurs after performing the clustering process, the DL-Tree has not to be entirely re-computed. Indeed, the similarity value between the new service description and all available service descriptions (leaf nodes of the DL-Tree) can be computed and the most similar available service is selected. Hence the new service description can be added as sibling node of the most similar service while the parent node is re-computed as the GCS of the old child nodes plus the new child. In the same way all the ancestor nodes of the new generated parent node are computed. In this way a single path of the DL-Tree is updated rather than the entire tree structure. In Fig. 4 an example of updating of the DL-Tree illustrated in Fig. 3 is reported, in case of a new service description **Z** occurs. Obviously, after a certain number of changing of the DL-Tree, the clustering process have to be recomputed.

## 5 Resource Retrieval exploiting DL-Tree

Resource retrieval is performed by matching a given request with available resource descriptions. In this section we present an algorithm that, by exploiting the DL-Tree returned by DL-Link algorithm (see Sect. 4), is able to accelerate the resource retrieval task as well as to outperform the service discovery task. The rationale of the algorithm is to cut the search space in order to drastically reduce the number of necessary comparisons (matches) for finding resources satisfying a given request. The algorithm is presented in the following.

Let $R$ be a request and let $C$ the root of the DL-Tree $\mathcal{C}$

**retrievalProcedure**$(R, C)$

1. returnedResource := null
2. if **matchTest**$(R, C)$ = false then
    – return returnedResource
3. else if C is leaf node then
    – returnedResource.add($C$)
4. else
    (a) if $C$ has left child node $C_l$ then
        i. returnedLeftResource = **retrievalProcedure**$(R, C_l)$
        ii. if returnedLeftResource != null then
            – returnedResource.add(returnedLeftResource)
    (b) if $C$ has right child node $C_r$ then
        i. returnedRightResource = **retrievalProcedure**$(R, C_r)$
        ii. if returnedRightResource != null then
            – returnedResource.add(returnedRightResource)
5. return returnedResource

**matchTest**$(R, C)$

1. if $(R \sqsubseteq C)$ then
    – return *true*
2. else
    – return *false*

The **retrievalProcedure** is performed by exploiting the DL-Tree whose leaf nodes are the available resource descriptions and every inner node (included the root node) is an intensional description of the child nodes. Given a request $R$, it is matched with the root node of the DL-Tree, in order to test if it can be satisfied by the available resources or not. Indeed, since the root node is the intensional description of all available resources, then if the match test is not satisfied (false returned value), this means that there are no available resources able to satisfy the request. On the contrary, if the test is satisfied, the match test is performed for each child node. If a child node does not satisfy the match subsumption condition, then the branch rooted in this node is discarded. Otherwise all
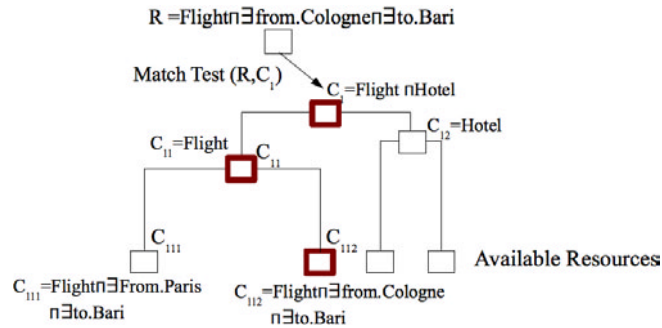
**Fig. 5.** Retrieval of an available service able to satisfy the request *R*. Bold boxes represent nodes satisfying the *MatchTest*.

the children nodes are recursively explored, until a leaf node is reached or until there are no child nodes satisfying the match condition.

As *MatchTest*, the *Entailment of Concept Subsumption* [16, 22] is used. It checks for subsumption, either of the requestor's description by the provider's or vice versa. Specifically, we check for the subsumption of description w.r.t. the request since we are interested in resources able to fully satisfy the request, while considering the vice versa, resources that only partially satisfy the request could be found.

It is important to note that, at the same level, more than one node could satisfy the match test. In this case all paths rooted in such nodes will be explored. As an alternative to such an exhaustive search, an heuristic could be adopted for suggesting the path to follow. As a possible heuristic, the path rooted in the node that is most similar to the request can be chosen. Differently from the previous case, in this way only one resource will be found, but it will be probably the most proper one.

The proposed method drastically reduces the size of the search space, since the search is restricted only to the branches of the DL-Tree whose nodes satisfy the match test. A good clustering of $n$ available resources (i.e. a balanced hierarchy) may reduce the number of necessary matches for finding the right resource from $O(n)$ to $O(log\ n)$ (if the request is specific enough), thus strongly outperforming the response time for a request. An example of application of *retrievalProcedure* procedure is reported in Fig. 5.

Here we conclude with the following lemma.

**Lemma 1.** *Resource retrieval using DL-Link algorithm and DL-Tree indexing is sound and complete.*

## 6 Experimental Evaluation

In this section some preliminary experiments are illustrated in order to show that the method proposed in Sect. 5 really improves the resource retrieval task w.r.t. to the traditional approaches based on linear matching.

### 6.1 Data sets

For measuring the efficiency of the retrieval method, it has been applied to the SE-MANTIC WEB SERVICE DISCOVERY DATA SET (SWS DISCOVERY DATA SET for brevity). It is a set of Semantic Web Services described by means of the DL-based framework presented in [10]. Since no existing data sets using such a framework were available, we have created a new one. Moving from the experience of OWLS-TC[10] that is a service retrieval test collection consisting of services specified by OWL-S 1.1, the SWS DISCOVERY DATA SET has been built. It consists of an $\mathcal{ALC}$ ontology representing the knowledge base of reference and a set of $\mathcal{ALE}(\mathcal{T})$ services described using such an ontology. The ontology models broad domains: bank domain, post domain, means of communication domain and geographical information. On the ground of such an ontology, 96 complex concept descriptions acting as service descriptions have been built. The resulting test set can be downloadable from https://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Projects/xmedia/dl-tree

### 6.2 Evaluation Methodology

The available service descriptions in SWS DISCOVERY DATA SET has been clustered by means of the DL-Link algorithm (see Sect. 4), obtaining as result a DL-Tree having, after the flattening post-processing, a depth equal to 7. This represents the maximum depth, while most of the leaf nodes in the obtained DL-Tree are at level $4$. The average branching factor is equal to $5$. In order to measure the efficiency of the retrieval method, different kinds of queries (requests) have been generated: 1) $\#\mathbf{96}$ queries corresponding to the leaf nodes of the tree (the actual service descriptions); 2) $\#\mathbf{20}$ queries corresponding to some inner node descriptions of the tree; 3) $\#\mathbf{116}$ queries randomly generated by disjunctions/conjunctions of primitive and/or defined concepts of the reference ontology and/or available service descriptions.

The efficiency of the method has been measured by counting the number of necessary comparisons (matches) in the tree for finding all available resources able to satisfy a request. Specifically, the average number of matches for each kind of query has been considered as well as the minimum and maximum number of matches. These values have been then compared with the number of checks in the linear retrieval case. Moreover, in both approaches, the average execution time for each kind of query has been measured, where the experiments have been performed on a laptop PowerBook G4 1.67 GHz 1.5 GB RAM.

### 6.3 Evaluation Results

Considering the queries generated as explained in the previous section and the DL-Tree obtained by clustering the SWS SERVICE DISCOVERY DATA SET, the retrieval procedure presented in Sect. 5 has been applied. The mean number of matches (subsumption checks) and the retrieval execution time have been measured. Specifically, for each query, the number of nodes visited (namely to which the subsumption check has

---

[10] http://projects.semwebcentral.org/projects/owls-tc/

**Table 1.** Number of comparison (average and range) and mean execution time for finding all the services satisfying a request w.r.t. the different kinds of requests both in the linear matching and in the DL-Tree based retrieval.

| DATA SET | Algorithm | Metrics | Leaf Node | Inner Node | Random Query |
|---|---|---|---|---|---|
| | **DL-Tree based** | *avg.* | **41.4** | **23.8** | **40.3** |
| | | *range* | 13 - 56 | 19 - 27 | 19 - 79 |
| SWS DISCOVERY DATA SET | | *avg. exec. time* | 266.4 ms. | 180.2 ms. | 483.5 ms. |
| | **Linear** | *avg.* | **96** | **96** | **96** |
| | | *avg. exec. time* | 678.2 ms. | 532.5 ms. | 1589.3 ms. |

been applied) for finding all the services able to satisfy a request has been counted as well as the execution time has been measured. Hence the mean number w.r.t. all queries for a given kind (randomly, inner node, leaf node) has been computed.

The outcomes of the experiments are reported in Tab. 1. Looking at the table it is straightforward to note the our method requires a very low number of matches w.r.t. the linear approach. Specifically, independently to the kind of considered request, the DL-Tree based retrieval decreases the number of comparisons more than 50%. Since a lower number of comparisons means a decreasing of the response time, then our method is really able to improve the efficiency of the retrieval and discovery process. This is also evident looking at the average execution time of the two methods. Focussing on the experimental result and specifically on the mean execution time for the different kinds of queries, it is possible to note that querying for inner nodes requires the lowest execution time. This is because most of the resources satisfying the queries are at the highest levels of the DL-Tree, so finding them requires less execution time. Moreover the structure of such queries is very simple. Since the adopted match test is based on subsumption test (see Sect. 5), consequently the time necessary for checking for subsumption decreases when the query is simple. Conversely, querying for randomly generated queries requires the highest execution time. This is because the structures of such queries are more complex then those of leaf node queries and inner node queries (that are the simplest ones). Particularly, the benefits of our approach increase with the increasing of available services.

From the presented initial experiments it is possible to assert that our method really improve the efficiency of the resource retrieval task.

## 7 Related Work

Service discovery is the task of locating service providers that can satisfy the requester's needs. Generally, it is performed by matching a request against available service representations - implying linear query performance.

Most of the current works concentrating on the automation of the service discovery task, focus on the improvement of the effectiveness of the service matchmaking, i.e. on engineering the service description. Central to the majority of the current SWS matchmaking approaches is that the formal semantics of service descriptions is explicitly defined in an ontology language such as OWL-DL [12]. In this way, service matchmaking can be performed by exploiting standard DL inferences [16, 22, 7, 13] sometimes jointly with the use of syntactic similarity measures [14].

Less attention has been dedicated to the improvement of the efficiency of the service discovery task, that, on the contrary, is the focus of our work. In [21], a way for turning efficient resource retrieval is proposed, by abstracting from a more expressive to a less expressive language, e.g. from OWL to DL-lite. Even if this approach is semantically sound, differently from our method, it looses completeness.

Most of the efforts have been employed for the optimization of reasoning and query answering. In [11], a set of optimization techniques for improving tableaux decision procedures for DLs are presented. They can be effectively used for performing service matchmaking. In [20], an algorithm for optimizing query answering of $\mathcal{SHIQ}$ knowledge bases extended with DL-safe rules is proposed, by exploiting the reduction to disjunctive programs. A combination of DL-tree retrieval with the optimizations defined in [20] could be more helpful than either on its own. Möller et al. [19] propose optimization techniques for improving the scalability of the instance retrieval task. This is orthogonal to our work as our method could be used also for performing instance retrieval by firstly clustering the MSCs of the considered knowledge base and then querying for the concept of interest, by checking for nodes of the DL-Tree that are subsumed by the query concept until leave nodes are found.

Another recent approach aiming at a scalable discovery process is *Semantic Discovery Caching* (SDC) [25]. It is based on an index structure, the SDC graph, that is a subsumption hierarchy made up of goal templates and usable Web services. Templates are organized w.r.t. their semantic similarity. The lower layer is the cache that captures knowledge on the usability of the available Web services. Based on this structure, the discovery process uses inference rules between the similarity degree of goal templates and the usability degree of Web services.

## 8  Conclusion and Future Work

We have presented a sound and complete method for improving the efficiency of the resource retrieval task and its validity has been experimentally shown. It is based on the exploitation of a tree-index (DL-Tree) that is built by applying a new conceptual clustering algorithm to available resource descriptions. For clustering resources, a new semantic similarity measure has been presented, while intentional cluster descriptions are built exploiting the Good Common Subsumer for $\mathcal{ALE}(\mathcal{T})$ concept descriptions.

Further work for improving the effectiveness of the similarity measure has been planned. Moreover, the validity of the method applied using different matching procedures will be verified. Furthermore, an incremental clustering algorithm will be developed, in order to cope with new available services avoiding the recomputation of a new clustering.

Our approach has been restricted to $\mathcal{ALE}(\mathcal{T})$ concept expressions and instances. However, the DL-Tree indexing procedure actually works with approximations: non-standard inference procedure like the Good Common Subsumer and the Most-Specific Concept. By approximating more expressive DLs expressions (i.e. OWL-DL expressions) to weaker languages, such as $\mathcal{ALE}(\mathcal{T})$ and $\mathcal{ALC}$, it is still possible to use the DL-Tree indexing procedure. Actual empirical evaluations of whether typical OWL-DL ontologies benefit from DL-tree indexing will however still have to be performed.

# References

1. F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

2. F. Baader, R. Küsters, and R. Molitor. Computing least common subsumers in description logics with existential restrictions. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 96–101. Morgan Kaufmann, 1999.

3. F. Baader, R. Sertkaya, and Y. Turhan. Computing least common subsumers w.r.t. a background terminology. In V. Haarslev and R. Möller, editors, *Proceedings of Proceedings of the 2004 International Workshop on Description Logics (DL2004)*. CEUR-WS.org, 2004.

4. H.H. Bock. *Analysis of Symbolic Data: Exploratory Methods for Extracting Statistical Information from Complex Data*. Springer-Verlag, 1999.

5. A. Borgida, T. Walsh, and H. Hirsh. Towards measuring similarity in description logics. In I. Horrocks, U. Sattler, and F. Wolter, editors, *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, volume 147 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2005.

6. M. W. Bright, A. R. Hurson, and Simin H. Pakzad. Automated resolution of semantic heterogeneity in multidatabases. *ACM Transaction on Database Systems*, 19(2):212–253, 1994.

7. S. Colucci, T. D. Noia, E. Di Sciascio, F. Donini, and M. Mongiello. Concept abduction and contraction for semantic-based discovery of matches and negotiation spaces in an e-marketplace. In *In Proc. 6th Int. Conference on Electronic Commerce (ICEC 2004)*. ACM Press, 2004.

8. C. d'Amato, N. Fanizzi, and F. Esposito. A semantic similarity measure for expressive description logics. In A. Pettorossi, editor, *Proceedings of Convegno Italiano di Logica Computazionale, CILC05*, Rome, Italy, 2005. `http://www.disp.uniroma2.it/CILC2005/downloads/papers/15.dAmato_CILC05.pdf`.

9. C. d'Amato, N. Fanizzi, and F. Esposito. A dissimilarity measure for $\mathcal{ALC}$ concept descriptions. In *Proc. of the 21st Annual ACM Symposium of Applied Computing, SAC2006*, 2006.

10. S. Grimm, B. Motik, and C. Preist. Variance in e-business service discovery. In *Proceedings of the ISWC Workshop on Semantic Web Services*, 2004.

11. I. Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

12. I. Horrocks, P. Patel-Schneider, and F. van Harmelen. From shiq and rdf to owl: The making of a web ontology language. *Web Semantics*, 1(1), 2004.

13. U. Keller, R. Lara, H. Lausen, A. Polleres, and D. Fensel. Automatic location of services. In *In Proceedings of the 2nd European Semantic Web Conference (ESWC 2005)*, volume 3532 of *LNCS*, 2005.

14. M. Klusch, B. Fries, and K. Sycara. Automated semantic web service discovery with owls-mx. In *AAMAS '06: Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*, pages 915–922, New York, NY, USA, 2006. ACM Press.

15. J. Lee, M. Kim, and Y. Lee. Information retrieval based on conceptual distance in is-a hierarchies. *Journal of Documentation*, 2(49):188–207, 1993.

16. L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 331–339, New York, NY, USA, 2003. ACM Press.

17. T. Mantay. Commonality-based ABox retrieval. Technical Report FBI-HH-M-291/2000, Department of Computer Science, University of Hamburg, Germany, 2000.

18. D. Maynard, W. Peters, and Y. Li. Metrics for evaluation of ontology-based information extraction. In *Proceeding of the EON 2006 Workshop*, 2006.

19. R. Möller, V. Haarslev, and M. Wessel. On the scalability of description logic instance retrieval. In In B. Parsia, U. Sattler, and D. Toman (Eds.), editors, *Proceedings of the International Workshop on Description Logics (DL2006)*, volume 189. CEUR, 2006.

20. B. Motik, U. Sattler, and R. Studer. Query answering for owl-dl with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

21. J. Z. Pan, E. Thomas, and D. Sleeman. Ontosearch2: Searching and querying web ontologies. In *Proc. of the IADIS International Conference*, 2006.

22. M. Paolucci, T. Kawamura, T.R. Payne, and K.P. Sycara. Semantic matching of web services capabilities. In *International Semantic Web Conference*, pages 333–347, 2002.

23. R. Rada, H. Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on System, Man, and Cybernetics*, 19(1):17–30, 1989.

24. P. Resnik. Semantic similarity in a taxonomy: An information-based measure and its application to problems of ambiguity in natural language. *Journal of Artificial Intelligence Research*, 11:95–130, 1999.

25. M. Stollberg and M. Hepp. Semantic discovery caching (sdc): Prototype and use case evaluation. Technical report, DERI, 07 April 2007.

26. E. M. Voorhees. Implementing agglomerative hierarchical clustering algorithms for use in document retrieval. *Information Processing and Management*, 22(6):465–476, 1986.