A Distributed Directory System

Fausto Giunchiglia and Alethia Hume

Department of Information Engineering and Computer Science
University of Trento, Italy
{fausto,hume}@disi.unitn.it
http://www.disi.unitn.it

Abstract. We see the local content from peers organized in directories (i.e., on local ordered lists) of local representations of entities from the real world (e.g., persons, locations, events). Different local representations can give different "versions" of the same real world entity and use different names to refer to it (e.g., George Lombardi, Lombardi G., Prof. Lombardi, Dad). Although the data from these directories are related and could complement each other, there are no links that allow peers to share and search across them. We propose a Distributed Directory System that constructs these connecting links and allows peers to: (i) maintain their data locally and (ii) find the different versions of a real world entity based on any name used in the network. We evaluate the approach in networks of different sizes using PlanetLab and we show that the results are promising in terms of the scalability.

Keywords: Name-Based Entity Search, P2P, Entity Directory

1 Introduction

We see Internet as a network of peers (a P2P network) organizing their content in directories, which digitally represent their own versions of *entities* that exist in the real world. Entities can be of different types (e.g., person, location, event and others), they have a name, and are described by attributes (e.g., latitude-longitude, size, birth date), which are different for different entity types [1]. Different versions of an entity can represent different points of view, they could show different aspects of the entity or the same aspects with different level of details. In a way, the local representations from peers can be seen as pieces of information about a particular entity that are stored in a distributed manner in the network.

In this network, the different directories contain related data and, to some extent, they can complement each other. One problem that prevents us from exploiting the relation between these data is that there are no links connecting the local directories from peers. An effort to connect related data on the web is that of Linked Data[1], which allowed linking important datasets like, dbpedia, Freebase, DBLP, ACM, and others. Nevertheless, this approach leaves out of

[1] http://linkeddata.org/

the semantic web the individual users (i.e., simple normal peers) and the data from their local directories stored in personal devices (e.g., smart-phones, PDAs, notebooks, etc.). We propose building a distributed directory that constructs the connecting links among the local directories at this level, i.e., the level of simple peers with personal devices. It it important to note that the whole directory can be seen as another dataset, which could be included as another node in the Linked Data graph. In this way, the directory would become the bridge that allows simple peers to participate as part of the semantic web as opposed to act only as consumers of it.

As in any directory, a peer normally identifies and distinguishes an entity from others by means of names (e.g., George Lombardi, Trento, Italy, University of Trento), which play a different role from the other attributes because they are identifiers rather than descriptions [2]. The values of other types of attributes have a meaning that can be understood, e.g., by mapping them to concepts from a knowledge base, like WordNet[2]. Names, on the other hand, are strings that behave similarly to keywords. Real world entities can be called by multiple names as a consequence of variations and errors. Moreover, the set of names used in different local representations to identify the same real world entity can be different, at the same time that the sets of names used to identify different real world entities can overlap.

The approach we propose for a *Distributed Directory System (DDS)* incorporates the notion of a real world entity described by different local representations from peers. This notion is used to organize the references to the local representations in order to allow finding all the available information about entities. Our system offers two main features:

- First, it takes into consideration that multiple, possible different, names can be used to identify the same real world entity (e.g., George Lombardi vs. G. Lombardi and Italy vs. Italia).
- Second, it allows peers to have control over the privacy of their data because the *DDS* stores only the names of the entity and a link to the local representation.

As a result, any name that is used in some local representation to identify an entity can be used to find all the different versions of that entity that are stored in the network of peers.

The paper is structured as follows. Section 2 presents a motivating example that shows a world of related directories, while Section 3 formalizes the basic notions that link the different directories. In Section 4, we explain the name matching problem that arises when linking different directories. Then, a distributed entity directory is proposed in Section 5 and the algorithms to perform search in such directory are explained in Section 6. The implementation and the evaluation details are discussed in Section 7. Finally, the related works are discussed in Section 8 and the conclusions are presented in Section 9.

[2] `http://wordnet.princeton.edu/`

2 A World of Directories

Nowadays, most of the organization of our data is done in terms of directories. A well known and old example is the telephone book directory, used to organize address and phone numbers of people and companies. Newer forms of directories can be seen, for example, in contact lists, document directories, event directories (i.e., calendars or agendas) used by peers in current devices (e.g., computers, PDAs, smart-phones) to organize the local representation of entities of their interest. Moreover, the data from different directories (possibly from different peers) can be related. Different peers attending to the same event might store local representations of the event. Each of them might also have the contact information of the other peers attending to the event, e.g., a meeting.

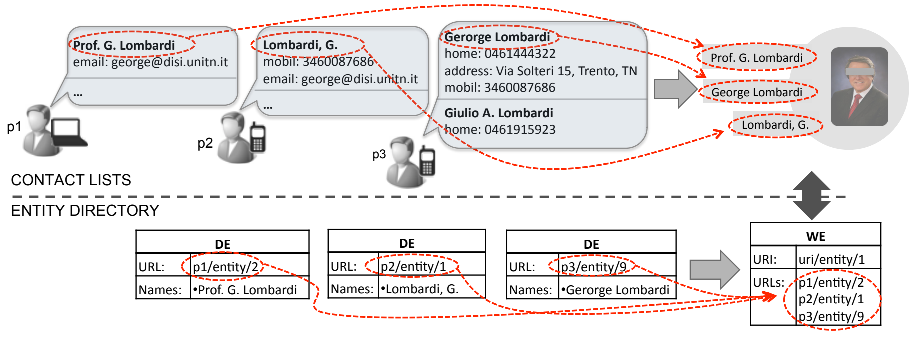

Fig. 1. Contact Lists Example

Let us consider in details the example of contact lists in different devices from the peers of a network that connects students, researchers and professors among them (e.g., SmartCampus[3]), and with their family members. The first part of Figure 1 (upper part) shows that the contact list of each device can be seen as a local directory of people. Different peers in this network can have different information about the people in their contact lists, like phone numbers, email addresses, skype user and others, which show different ways to get in touch with them. For example, suppose that p_1 is a student that is taking a course with prof. George Lombardi and therefore p_1 has, in its contact list, the university email address of the professor. A researcher p_2 that is working with him could have more information, like his email and mobile phone number. On the other hand, a family member p_3 may have his home address and phone number but not the university email (because such information is not relevant for p_3).

Now, suppose that another researcher in the network, let us call it p_4, hears about prof. Lombardi work and wants to contact him. We can see that:

[3] http://www.smartcampuslab.it

1. First, the information that p_4 needs is distributed in the network and the problem is knowing where the different pieces are stored
2. Second, the different peers can call the same person using different names, e.g., Prof. Lombardi, George Lombardi, G. Lombardi. In our example, this means that p_4 need to be sure that the other peers (i.e., p_1, p_2 and p_3) are all referring to the same person as he is.
3. Third, the contact information can change in time. The work email of Prof. Lombardi will change if his affiliation changes, his phone numbers can change at any time, and his address will change if he changes residence.
4. Finally, the privacy and the sensitiveness of the information have to be considered. Most likely the phone number and address of the home of prof. Lombardi would be more private than the university email. As a consequence, $p3$ will not share such information with everyone.

3 Linking Directories

We define a Directory of Entities that formalizes the links between data from different directories through the distinction between a Digital Entity (DE) and a Real World Entity (WE). A DE is defined as a local representation of an entity that exist in the real world. A URL (Uniform Resource Locator) is used in order to uniquely identify a DE and it can be used (by dereferencing) to obtain the full local description (i.e., based on attributes). We also consider a set of names $\{N\}$ as the human readable identifiers used in DEs to refer to a WE and distinguish it from others. Formally,

$$DE = \langle URL, \{N\} \rangle \tag{1}$$

On the other hand, a WE represents the real world entity and is modeled as a class of DEs. We use a URI (Uniform Resource Identifier) to uniquely identify each WE. Formally,

$$WE = \langle URI, \{URL\} \rangle \tag{2}$$

where $\{URL\}$ is a non-empty set of identifiers of different DEs that describe WE. As a consequence of the composition of these definitions we can see that multiple sets of names are given to a WE through DE definitions from different peers that describe the same WE.

In the second part of Figure 1 (lower part) we show how the example from Section 2 can be formalized in terms of these notions (i.e., DEs and WEs). We can see a *one-to-one* mapping between the WE from an entity directory and the real person represented in different contact lists. Moreover, we see that an entry from a contact list is translated into a DE in the directory (i.e., also a *one-to-one* mapping). There is a *one-to-many* relation between WEs and DEs which shows that each single entry in a contact list correspond to one person but one person can be described in many different entries (possibly from different peers). Finally, the relation between $Names$ and WEs introduces a name matching problem that is better discussed in the following section.

Note that these notions allow the separation between "what" is being represented and "where" is being represented. This separation is needed in order to model the issues stated in items 1 and 2 from the example of Section 2. The DEs model the different pieces of information that p_4 needs and their $URLs$ tell us where they are. The WE models the link that connects different DEs and its URI identify what they represented. Regarding item 2, we can see that different sets of names are given in DEs, which models the fact that p_1, p_2 and p_3 can define the different names that they use to call an entity.

On the other hand, the distinction between the two notions (DE and WE) also provide the infrastructure to deal with the issues introduced by the other two items (i.e., items 3 and 4 in Section 2). The dynamism of the information about the entities and the privacy of local data are constrained to affect DEs. In this way, when the email of Prof. Lombardi changes (see Figure 1), p_2 (the researcher) updates its local representation (i.e., the DE). The corresponding WE definition is not affected by this update, nevertheless the information (available in the P2P network) about Prof. George Lombardi is updated. Similarly, access control can be implemented over the data associated to each single DE representation, which do not affect WE definitions. Note that such implementation (i.e., access control implementation) is out of the scope of this paper, but the interested readers are invited to see (for example) [3].

4 Name Matching

Names are human readable identifiers that serve the purpose of distinguish an entity from others. They are labels composed by a combination of words, numbers and symbols [2]. In the context of our entity directory, we define the set of names that identify a WE as the union of the names used in DEs that locally represent that WE in different peers. Names are different from other attributes because they play the role of keywords rather than been mapped to concepts from a knowledge base. As such, names can suffer from different types of variations. Following the results from the study performed in [4], we can distinguish among the following types:

- **Format.** The format variations have a strong dependence with entity type and affect mostly to people names. They include the variation of the order in which the words of a name can be written (e.g., *George Lombardi* and *Lombardi, George*) and the multiple abbreviations that can exist for the same full name (e.g., *Giulio Augusto Lombardi* can be abbreviated as *G. A. Lombardi*, *Giulio A. Lombardi* and others). It is also important to notice that the abbreviation of a name can be a valid reference to many different full names (e.g., *G. Lombardi* is valid for *George Lombardi* but also for *Giulio Lombardi*).
- **Full translations.** Names sometimes are written differently in different languages (e.g., *Trento* in Italian, *Trient* in German or *Trent* in English).
- **Part-of translations.** In other cases only one part of the name changes in different languages. This is the case of names composed by common and

proper nouns, where the common noun is called trigger word in [4] and is the only part that is affected by the translation (e.g., *University of* Trento vs. *Università di* Trento).
– **Misspellings.** Names can be misspelled, either in the definition of a DE or during the specification of a search query. The misspellings can be a consequence of variations in the punctuation, capitalization, spacing, omissions, additions, substitutions, phonetic variations (e.g., *Fasuto* vs. *Fausto*, *G Lombardi* vs. *G. Lombardi*).
– **Pseudonyms.** Entities also have pseudonyms that are not (necessarily) variations of a name but rather alternative names for an entity, which can be defined (and used) in different contexts. This is the case for some arbitrary nicknames that are sometimes used by peers to refer to a DE (e.g., *Fede* is commonly used as a nickname for *Federico* or *Federica* and *The King of Rock and Roll* is a common nickname for *Elvis Presley*).

The name variations together with the DE definition presented above, show that the relation between names and DEs is of the type *many-to-many.* In turn, this leads to a name-matching problem when we intend to search an entity based on its names [2]. This problem, in the context of the entity directory, can be decomposed in:

1. The problem of matching names inside the network: A name used in a DE can be a variation of the name used in another DE that represent the same WE. We need to take into consideration all the multiple names (including name variations) used in the network to identify a WE and match them to all the different DEs that describe WE. In the example from Figure 1, if the user is searching an entity with the name *"George Lombardi"*, the directory should be able to return all the DEs (i.e, *p1/entity/2*, *p2/entity/1* and *p3/entity/9*) that represent the different versions of *uri/entity/1* rather than only returning the one that give it such name (i.e., *p3/entity/9*).
2. The problem of matching queries with the names used in the network: This case considers query names that are unknown to the entity directory, but that are however variations of one or more known names. We say that a name is unknown to the directory if there is no DE in the network that uses such name to identify a WE. The easiest example is a query name that is misspelled with regard to the DEs of the directory. In the example from Figure 1, if the user input the query *"Goerge Lombardi"*, the search should be able to find that *"George Lombardi"* is a candidate match.

5 A Distributed Directory System

In this paper we propose a Distributed Directory System (DDS) that organizes information about entities incorporating the notions of WE and DE, which were presented in Section 3. These notions allow the separation of the problem of finding the DEs that represent different versions of a WE from the problem of finding WEs that are identified with multiple names. We exploit this separation by building two different indexes, one to deal with each problem.

A *DEindex* is created to map WEs (i.e., $URIs$) to DEs (i.e., $URLs$) and can be formally defined as,

$$DEindex = \{WE \rightarrow DE \mid \nexists WE' \rightarrow DE \in DEindex\ s.t., WE' \neq WE\} \quad (3)$$

We can see that this index encodes the *one-to-many* relation between WEs and DEs because the mapping of different WEs to the same DE is not allowed. On the other hand, a *WEindex* is created to map the names that are given (in local representations) to WEs (i.e., $URIs$). Let us call $\{N^{DE}\}$ to the set of names of a digital entity DE. Then, the *WEindex* can be formally defined as,

$$WEindex = \{N \rightarrow WE \mid \exists WE \rightarrow DE \in DEindex\ s.t., N \in \{N^{DE}\}\} \quad (4)$$

We can see that this index encodes the *many-to-many* relation between $Names$ and WEs because the only constraint on the mappings is related to the existence of a local representation that gives "support" to such mapping.

Let us now discuss in more details how the publication, maintenance and search of entities are done in the DDS:

The *publication and deletion of* DEs in the network are the two main events that modify the DDS by affecting the content of the indexes defined above. The publication of a DE affects both indexes in a straightforward manner. First, the DE is associated to the WE that it represents by adding the corresponding mapping (i.e., $WE \rightarrow DE$) to the $DEindex$. Second, the mappings $N_i^{DE} \rightarrow WE$, of each name N_i^{DE} in $\{N^{DE}\}$ to the WE that is associated to the DE, are added to the $WEindex$. In order to do this, we assume that the peer locally caches the identifier (i.e., the URI) of the WE that is represented by its DE[4]. On the other hand, when a DE is deleted from the network, only the $DEindex$ is directly affected. The same mapping $WE \rightarrow DE$ that is added when the DE is published, is then removed from the $DEindex$ when the peer deletes the DE. Regarding the $WEindex$, we say that it is not directly affected because the mappings of names can be removed only after verifying that they are no longer valid to identify the corresponding WE. Such verification is further discussed as part of the DDS maintenance.

The *maintenance* of the DDS is performed through periodic checks over the indexes in order to detect and remove entries that are no longer valid. In the $DEindex$, an entry can be considered invalid if it contains mapping to a DE that has been unreachable for a long time. In order to detect this situation, each entry is attached with a timestamp corresponding to the last time when the DE was reachable. This timestamp is updated in every periodic check. When the DE is not reachable, the interval between the last reachable time and the current time is verified. The corresponding entry is removed from the $DEindex$ if such interval exceeds a given threshold. An entry from the $WEindex$, on the

[4] Note that the initial identification of the WE described by a DE is a problem of identity management and is out of the scope of this work. See for example [5, 6]

other hand, is considered invalid if it contains a mapping that do not complies with the constraint established by the index definition presented in equation 4. This means that a mapping between N and WE has to be removed from the $WEindex$ when there are no DEs in the network using the name N to refer to such WE. In other words, when none of the available entities provide support to such mapping.

Search in the DDS can be performed using two different types of identifiers, $URIs$ and *names*. In this context, having as input a URI means that the target WE has been uniquely and fully identified. Therefore, the goal of the search is to obtain all the different representations (i.e., the DEs) of the WE. On the other hand, in a search based on names, we need to find the candidates WEs (to be the right answer) as a consequence of the *many-to-many* relation between names and WEs. After the candidates WEs has been found, we can use the search by URI to find the different representations of them. In what follows, the search by names is considered in more details while the search by $URIs$ is included as a part of the former.

A query is formally defined as $Q = \{N^Q\}$, where $\{N^Q\}$ is the non-empty set of names used to identify one target WE. Then, the problem of searching entities based on their names can be seen as retrieving WEs that are described in the network by at least one DE, such that, the intersection between $\{N^{DE}\}$ and $\{N^Q\}$ is not empty. This definition considers a partial matching between $\{N^{DE}\}$ and $\{N^Q\}$ in order to allow finding a WE from any of the names given to it on different DEs. In turn, this can be translated in the formal definition of the Query Answer (QA) as follows:

$$\begin{aligned} QA = \{\langle WE, \{DE\}\rangle \mid \exists N' \in \{N^Q\} : N' \rightarrow WE \ \in \ WEindex \\ \wedge \ \forall DE' \in \{DE\} : WE \rightarrow DE' \in DEindex\} \end{aligned} \tag{5}$$

As we mentioned before, this answer is build in two steps. The algorithms that perform the two steps are presented in Section 6.

6 Algorithms

We assume that the indexes offer non-blocking APIs (to allow the parallelization of index lookups), which mean that a call to the *GET* function on the indexes returns immediately a reference to an object that will be filled with the results from the index lookup. In Algorithm 1, we define the global data structures, which are strictly related to the indexes. They are used across the different functions involved in the search. We use the statement *for all* (line 6 in Algorithm 2 and line 8 in Algorithm 3) to denote the concurrent execution of the statements that are in its body (i.e., line 7 in Algorithm 2 and lines 9 to 24 in Algorithm 3).

The Search Entity function is presented in Algorithm 2 and is the main entry point for the search by names. This function receives the query names and returns a set of candidate WEs according to the constraints given in Equation 5. In order to measure how relevant each candidate WE is, we count the number of query names that match with the names associated to the WE. This relevance is associated to each candidate WE and included in the resultset. In line 7, the first step of the search by names is initiated with the call to the $GetWEindex$ function of the $WEindex$. The object returned by the function is given to the corresponding handler function, which knows how to process it.

Algorithm 1 Global Data Structures

```
1: WEAnswer : ⟨isComplete, name, weAnsValues⟩
2: DEAnswer : ⟨isComplete, URI, deAnsValues⟩
3: isComplete : boolean                      ▷ TRUE when the index lookup is finished
4: weAnsValues : NULL OR {URI} OR {URL} OR {{URI} ∪ {URL}}
5: deAnsValues : {URL}                                   ▷ not empty set of URLs
```

Algorithm 2 Search Entity

```
1: function SearchEntity(names : {name}) → {⟨WE, relevance⟩}
2:     WEs : {⟨WE, relevance⟩}                          ▷ stores search results
3:     WE : ⟨URI, {URL}⟩                    ▷ {URL}.size == 1 when URI == NULL
4:     relevance : integer
5:     WEs := {}
6:     for all name ∈ names do                              ▷ Parallel threads
7:         HandleWEAnswer(GetWEindex(name), WEs)
8:     end for
9:     return WEs
10: end function
```

The Algorithm 3 shows the $HandleWEAnswer$ function, which is in charge of processing the values retrieved from the $WEindex$. We can see from lines 4 to 6 the loop that waits until the answer is completed. Then, in line 8, we start one execution thread to process each retrieved value. A value returned from the $WEindex$ represents a WE, it can be a URI or a URL (see line 4 from Algorithm 1). In the former case, we say that the WE identity is known. The corresponding instance is created (line 10 in Algorithm 3) with the global identifier and an (up to now) empty set of DEs. In the later case, the URL identifies a WE with no global identifier and we assume that there is only one DE that describes it (line 18 in Algorithm 3).

In lines 11 and 19, we check whether the WE is already in the result-set. If it is, we call the function *relevanceWE++*, which increments the count of the relevance that is associated with the WE. Otherwise, we *add* the WE to the result-set with a relevance count initiated to 1 (lines 14 and 22). At this point, if we are in the case of a WE with global identifier (i.e., with a URI), the second step of the search is initiated with the call to the $GetDEindex$ function of the *DEindex* (see line 15). The object returned by the function is given to the *HandleDEAnswer* function, which then process it.

Algorithm 3 Handler of the WE Answers

```
 1: function HandleWEAnswer(weAnswer : WEAnswer, WEs : {⟨WE, relevance⟩})
 2:     waitingTime : integer
 3:     waitingTime := 5                                 ▷ parameterizable waiting time
 4:     while weAnswer.isComplete = FALSE do
 5:         WAITms(waitingTime)                               ▷ specified in milliseconds
 6:     end while
 7:     if weAnswer.weAnsValues $\neq$ NULL then
 8:         for all weAnsValue $\in$ weAnswer.weAnsValues do          ▷ Parallel threads
 9:             if isURI(weAnsValue) then
10:                 wEntity := ⟨weAnsValue,{}⟩
11:                 if wEntity $\in$ WEs then
12:                     relevanceWE++(WEs, wEntity)
13:                 else
14:                     add(WEs,⟨wEntity,1⟩)
15:                     HandleDEAnswer(GetDEindex(weAnsValue), WEs)
16:                 end if
17:             else
18:                 wEntity := ⟨NULL,{weAnsValue}⟩
19:                 if wEntity $\in$ WEs then
20:                     relevanceWE++(WEs, wEntity)
21:                 else
22:                     add(WEs, ⟨wEntity,1⟩)
23:                 end if
24:             end if
25:         end for
26:     end if
27: end function
```

Algorithm 4 Handler of the DE Answers

```
1: function HandleDEAnswer(deAnswer : DEAnswer, WEs : {⟨WE, relevance⟩})
2:     waitingTime : integer
3:     waitingTime := 5
4:     while deAnswer.isComplete = FALSE do
5:         WAITms(waitingTime)
6:     end while
7:     addDE2WE(WEs, deAnswer.key, deAnswer.deAnsValues)
8: end function
```

Finally, the Algorithm 4 shows how the values retrieved from the $DEindex$ are handled. First, we wait until the answer is completed (see the loop from line 4 to line 6) and then the values are used to update the resultset. Note that the function *addDE2WE* takes the key (i.e., the URI) to identify, in the resultset, the WE that has to be updated. The values (i.e., the $URLs$) are then associated to such WE in order to complete the QA. We say that this function (called in line 7 in Algorithm 4) adds DEs to a given WE from a given set.

7 Implementation and Evaluation

We implement the distributed directory on top of a P2P network, where the distribution of the indexes is done using a Distributed Hash Table (DHT). DHTs[5] allow the peers participating in the network to store and retrieve pairs of key and value. In particular, we use TomP2P[6], an advanced DHT library that extends the basic functions of DHTs. The library supports storing multiple values mapped to the same key and distinguishes between different index domains. The execution of the operations over different index domains can be seen as having different DHTs, i.e., one for the $DEindex$ and other for the $WEindex$.

We are interested in the evaluation of the approach under realistic network conditions and we want to measure how much the performance decreases when the size of the network grows (i.e., the scalability). The performance is considered here in terms of the time that takes the system to process a query. We use *PlanetLab*[7] as a testbed because we believe it gives us the realistic network conditions that we need. PlanetLab provides a network of computers (i.e., nodes) that are distributed around the world, connect to each other through the internet and are available for research purposes. We perform the evaluations on networks of 50, 100 and 150 peers and the data extracted from the proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)[8] are used to generate the data-sets. We use the titles of publications, names of authors and names of locations related to the conference.

Each data-set is produced by generating triples of $\langle Name, URI, URL\rangle$. The names and $URIs$ are replicated in order to simulate different WEs having the same name and different peers storing DEs that describe the same WE. Let us call p_n to the popularity of a name n (i.e., number of WEs that are called by n) and p_{we} to the popularity of a WE (i.e., number of DEs that represent WE). First, for each name n, we generate p_n triples with the same name (different URI and URL). Second, for each URI, we generate p_{we} triples with the same name and URI but with different $URLs$. The popularities p_n and p_{we} follow a Zipf[9] distribution, which means that there is a long tail of unpopular names and WEs. The distribution of both popularities are independent, which means

[5] `http://en.wikipedia.org/wiki/Distributed_hash_table`
[6] `http://www.tomp2p.net/`
[7] `https://www.planet-lab.eu/`
[8] `http://ijcai.org/`
[9] `http://en.wikipedia.org/wiki/Zipf's_law`

that a popular WE do not necessarily has a popular name and vice versa. We assume that the local entity base of each peer contains, in average, 2000 DEs. We have overall around 100000, 200000 and 300000 DEs. The query set for each peer is generated by randomly selecting a set of 1400 names from the initial set of entity names.

During the evaluation, we first index the data-set for the corresponding network size and then the peers begin the search evaluation process pseudo-simultaneously. In this process, each peer performs the following steps: (i) takes a query from the query set, (ii) runs the search, (iii) measures and logs the time that the system takes to respond to the query, (iv) waits a random interval of time (between 1 and 3 seconds), and (v) go back to step (i). These steps are repeated until the end of the set of queries. Once all the peers end the search process, we compute the average query time for the network. We show the results for the different network sizes in Table 1. The values for the average query times

Table 1. Average query time

Network Size	50 peers	100 peers	150 peers
Avg. Query Time (in seconds)	2.77	2.75	2.61

are stable with the network growth and we believe this is a promising result regarding the scalability of the directory. On the other hand, when comparing to information retrieval systems (in general), the average times for search are still high.

In order to have better understanding of the query times that contribute to these averages, we analyze the distribution of the query time in the different networks. In Figure 2 we show the results of this analysis, where we can see that also the query time distribution is stable with regard to the network growth. Also in Figure 2 we can notice that more than 55% of the queries are actually answered in less than a second, while in almost 70% of the cases the response arrives in less than 2 seconds (which is less than the average time). Moreover, only 9% of queries take more than 5 seconds to be answered.

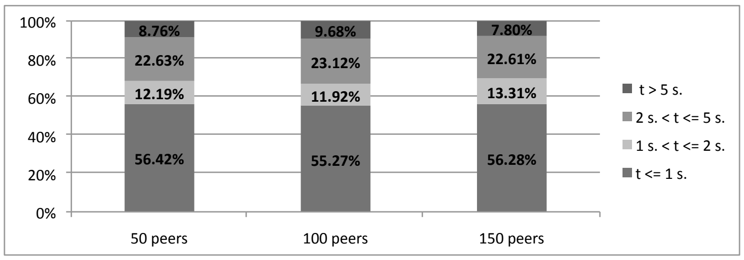

Fig. 2. Query time of different networks

It has to be noted that the results are returned after the query answer is complete, i.e., once all the lookups involved in the query have ended. This means that a single slow lookup is enough to delay the computation of a query answer and therefore increase the query time. Furthermore, we know that particularly slow peers can produce this problem when a lookup has to be routed through them. We believe that, in the big picture, the scalability of the approach is a promising and important result. On the other hand, there are some techniques to perform result catching or to avoid routing through slow peers (see for example [7]) that can be implemented to reduce the effect of slow peers at query time.

8 Related Work

The work introduced in this paper involve the approaches that are capable of managing information about entities in a P2P network. More specifically, our approach deals with the distributed indexing and searching of entities based on their identifiers. To the best of our knowledge there are no approaches that integrates these areas, i.e., that performs search of entities over a p2p network. Nevertheless, we give an overview of related approaches from both areas.

Some entity aware approaches concentrate the attention on the definition of models and structures for the representation of entities [1]. In [6] an entity name system (ENS) is proposed in order to provide support for the generation and reuse of globally unique identifiers for entities across different and independent RDF repositories. The local repository of a single user is not considered as a source of data and the users need a special access permit in order to contribute with the definition of entities. As a first step towards searching, the work presented in [8] proposes a model that analyzes the query specification and performs the disambiguation of the desired type of entity. In [9], named entities are extracted by analyzing queries based on syntactic matching of patterns. These approaches do not directly address the search, but their results are relevant for the definition of the directory proposed in this paper.

Other approaches that perform search following an entity centric perspective can be found in the literature [10–12]. Entity search engines are proposed in [10, 12], heuristic rules are used in [11] to identify entities appearing in a collection of documents and a service to find documents that contain statements about particular resources is provided in Sindice [13]. Most of these approach collect data from multiple web sources (i.e., by crawling) but do not consider distribution at the level of single users (i.e., a p2p network). In particular, [12] automatically aggregates descriptions from the different sources and allows subsequent navigation to related entities. Distribution is considered in terms of clusters of computers that allow parallel processing and scalable storage but the search is centralized (i.e., they build centralized indexes). In contrast to these approaches, our approach performs a distributed search in a P2P network and allows users to maintain their data locally.

On the other hand, we have P2P approaches, which perform distributed search but are not aware of entities [14, 15]. They are mainly classified as unstructured and structured approaches. The first unstructured networks (e.g., Gnutella[10]) have scalability problems due to the number of messages generated and do not guarantee that all answers will be found. Other approaches use clustering techniques [16–20], their goal is to find the best group to answer a query and then send the query to the peers in that group. Our approach can find all available answers and has proven to be promising in terms of scalability.

We can find also more structured approaches that aim to guarantee the location of the content shared on the network (e.g., CAN [21], Chord [22] Pastry [23] and Tapestry [24] They store pairs of $\langle key, value\rangle$ in a Distributed Hash Table (DHT) and then retrieve the value associated with a given key. Other approaches perform multi-keyword search using DHTs but they can be very expensive in terms of required storage and generated traffic (e.g., see [25]). Hierarchical structures combine clustering techniques with the structure of DHTs [26–29]. In general, P2P approaches provide the techniques needed in order to build our solution. The novelty of our approach is in the domain of application of such techniques.

9 Conclusions

We presented and approach for a distributed directory of entities that introduces the notions of DE and WE in order to link local directories of different peers. The directory provides search services based on entity identifiers. In particular, we presented the algorithms for searching entities based on their names. We discussed the name matching problem that appears as a consequence of the *many-to-many* relation between names and WEs. Then, we showed that, by its design, our directory deals with the problem of matching names inside the network (i.e., the first part of the name matching problem).

The data from peers are stored locally, only the identifiers and the links to the local representations are indexed. This infrastructure allows the implementation of access control mechanisms on the local representations in order to deal with privacy issues. At the same time, the changes made by peers in local representations, are available in the directory in a straightforward manner. The indexes are distributed using a Distributed Hash Table (DHT) but the directory definition is independent from a specific underlying DHT implementation.

The evaluation of the search was performed on networks of 50, 100, and 150 peers running on PlanetLab. The average query time (as a measure of the performance) for different network sizes were presented as well as the distribution of the query times. The results can be considered promising in terms of scalability because the performance is stable with the network growth.

As part of the future works, we want to study and integrate (possibly existing) approaches to deal with the problem of matching queries with the names used in the network (i.e., the second part of the naming problem). Additionally, we want to better understand the different elements that influence the search performance in order to find and implement techniques to reduce the query times.

[10] `http://en.wikipedia.org/wiki/Gnutella`

References

1. Bazzanella, B., Chaudhry, J.A., Themis Palpanas, Stoermer, H.: Towards a General Entity Representation Model. 5th Workshop on SWAP (2008)
2. Holloway, G., Dunkerley, M.: The Math, Myth and Magic of Name Search and Matching. 5th edn. Search Software America (2004)
3. Giunchiglia, F., Zhang, R., Crispo, B.: Relbac: Relation based access control. In: Proceedings of the 2008 Fourth International Conference on Semantics, Knowledge and Grid. SKG '08, Washington, DC, USA, IEEE Computer Society (2008) 3–11
4. Bignotti, E.: Semantic name matching. Master's thesis, University of Trento (2012)
5. Hogan, A., Zimmermann, A., Umbrich, J., Polleres, A., Decker, S.: Scalable and distributed methods for entity matching, consolidation and disambiguation over linked data corpora. JWS: Science, Services and Agents on the World Wide Web **10** (2012)
6. Bouquet, P., Stoermer, H., Niederee, C., Maña, A.: Entity name system: The backbone of an open and scalable web of data. In: Proceedings of the 2nd IEEE ICSC, Washington, DC, USA, IEEE Computer Society (2008) 554–561
7. Rhea, S., Chun, B.G., Kubiatowicz, J., Shenker, S.: Fixing the embarrassing slowness of opendht on planetlab. In: Proc. of the 2nd conference on Real, Large Distributed Systems. WORLDS'05, Berkeley, CA, USA (2005) 25–30
8. Bazzanella, B., Stoermer, H., Bouquet, P.: Searching for individual entities: a query analysis. Technical report, University of Trento (2009)
9. Paşca, M.: Weakly-supervised discovery of named entities using web search queries. In: Proceedings of the sixteenth ACM conference on CIKM '07, New York, NY, USA, ACM (2007) 683–690
10. Cheng, T., Chang, K.C.C.: Entity search engine: Towards agile best-effort information integration over the web. In: CIDR 2007. (2007) 108–113
11. Hu, G., Liu, J., Li, H., Cao, Y., Nie, J.Y., Gao, J.: A supervised learning approach to entity search. In: AIRS'06. Volume 4182 of LNCS. (2006) 54–66
12. Hogan, A., Harth, A., Umbrich, J., Kinsella, S., Polleres, A., Decker, S.: Searching and browsing linked data with swse: The semantic web search engine. JWS: Science, Services and Agents on the World Wide Web **9**(4) (2011) 365 – 401
13. Oren, E., Delbru, R., Catasta, M., Cyganiak, R., Stenzhorn, H., Tummarello, G.: Sindice. com: a document-oriented lookup index for open linked data. International Journal of Metadata, Semantics and Ontologies **3**(1) (2008) 37–52
14. Risson, J., Moors, T.: Survey of research towards robust peer-to-peer networks: Search methods. Computer Networks **50** (2006) 3485–3521
15. Lua, E.K., Crowcroft, J., Pias, M., Sharma, R., Lim, S.: A survey and comparison of peer-to-peer overlay network schemes. IEEE Communications Surveys and Tutorials **7** (2005) 72–93
16. Bawa, M., Manku, G., Raghavan, P.: Sets: Search enhanced by topic segmentation. In: Proceedings of ACM SIGIR Conference. (2003) 306–313
17. Cohen, E., Kaplan, H., Fiat, A.: Associative search in peer to peer networks: Harnessing latent semantics. In: Proceedings of IEEE INFOCOM. (2003)
18. Spripanidkulchai, K., Maggs, B., Zhang, H.: Efficient content location using interest-based locality in peer-to-peer systems. In: Proceedings of IEEE INFOCOM. Volume 3. (2003) 2166–2176
19. Crespo, A., Garcia-Molina, H.: Semantic overlay networks for p2p systems. Technical report, Stanford University (2002)

20. Joseph, S.: Neurogrid: Semantically routing queries in peer-to-peer networks. In: Proc. Intl. Workshop on Peer-to-Peer Computing. (2002) 202–214
21. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. In: Proc. of SIGCOMM'01, NY, USA, ACM (2001) 161–172
22. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: Proc. of SIGCOMM'01, NY, USA, ACM (2001) 149–160
23. Druschel, P., Rowstron, A.: Pastry: scalable, distributed object location and routing for large-scale peer-to-peer systems. In: Proc.of ACM SIGCOM. (2001)
24. Zhao, B., Huang, L., Stribling, J., Rhea, S., a.D. Joseph, Kubiatowicz, J.: Tapestry: A Resilient Global-Scale Overlay for Service Deployment. IEEE Journal on Selected Areas in Communications **22**(1) (January 2004) 41–53
25. Li, J., Thau, B., Joseph, L., Hellerstein, M., Kaashoek, M.F.: On the feasibility of peer-to-peer web indexing and search. In: IPTPS'03. (2003)
26. Ganesan, P., Gummadi, K., Garcia-Molina, H.: Canon in g major: designing dhts with hierarchical structure. In: ICDCS'04. (2004) 263 – 272
27. Janakiram, D., Giunchiglia, F., Haridas, H., Kharkevich, U.: Two-layered architecture for peer-to-peer concept search. In: 4th Int. Sem Search Workshop. (2011)
28. Papapetrou, O., Siberski, W., Nejdl, W.: Pcir: Combining dhts and peer clusters for efficient full-text p2p indexing. Computer Networks **54**(12) (2010) 2019–2040
29. Garcés-Erice, L., Biersack, E.W., Felber, P., Ross, K.W., Urvoy-Keller, G.: Hierarchical peer-to-peer systems. In: Euro-Par. (2003) 1230–1239