Creation of Semantic Overlay Networks Based on Personal Information

Alberto García-Sola and Juan A. Botia

University of Murcia
{agarciasola,juanbot}@um.es

Abstract. In P2P systems, nodes typically connect to a small set of random peers to query them, and they propagate those queries along their own connections. To improve that mechanism, Semantic Overlay Networks influence those connections depending on the content of the peers, clustering peers in overlapped groups (Semantic Overlay Networks). Ontologies are used for describing semantic information of shared items and user profile. Once the peers are grouped by their semantic information, we can take advantage of that distribution to add some new functionalities as recommendation. In this paper we focus on the description and evaluation of SONAtA, a SON classifier algorithm to globally organize peers into semantic groups, executed locally in each peer.

Keywords: SON, Semantic Web, Web 2.0, P2P.

1 Introduction

In the recent few years, P2P activity has vastly increased. Semantic Overlay Networks (SON) [1] organize peers from P2P networks into semantic groups in order to improve their efficiency. The main idea behind SON consists on creation and management of a flexible network organization, improving query performance based on the semantic relations among peers. In order to do that, the notion of cluster of peers is used. Peers arrange into clusters according to the content they share. Clusters may overlap, because peers can contain different content and belong to several clusters. In this context, queries are distributed to relevant clusters only and flooded among relevant peers at a cluster, reducing unnecessary traffic while making queries more efficient. In this paper, we introduce SONAtA, the SON classifier algorithm used to gather peers of such networks into semantic groups, so that they can interact with each other, enhancing their social capabilities. That gathering is the key-step to create and maintain SONs. A full description about the P2P system where SONAtA is placed can be found at [2].

The following requirements are key to be satisfied by a SON. SONs must have **small population of peers**. The smaller the number of peers we need to search, the better the query performance. A classification should not be too much specific, since too few peers would be in the SON, neither too general, in order to avoid a SON hosting the majority of peers in the system. In a good classification peers must have connections with **small number of SONs**, the greater the number of SONs, the greater the cost for a peer to keep track of all of them. Systems compound by SONs must be **tolerant to classification errors**. There are many sources of errors in items classification, like user wrong classification, fakes and other. A peer may be correctly classified even if some of its items are misclassified.

A lot of work related with Overlay Networks has been done. Crespo and Garcia-Molina [1,3] introduced the concept of Semantic Overlay Networks. We can find there the notion of classification hierarchy but it is not designed in the sense of a modern ontology based on RDF or OWL-DL languages. Tang and Xu use SONs to develop pSearch [4], a decentralized non-flooding P2P information retrieval system. Aberer and Cudre-Mauroux [5] address the problem of building scalable SONs proposing GridVine [6], a semantic overlay infrastructure based on a decentralized access structure. They use RDF/RDFS to encode meta-data and vocabulary definition. A similar approach is followed by INGA [7]. In all these works, semantic information is used for query routing and some for peer classification. Most of them do not describe how the SONs are actually created. And none of them uses personal information for, through the use of an ontology, create SONs in an autonomous and decentralized way.

The rest of the paper is structured as follows. Section 2 introduces SONAtA as a SON classifier. In 3 we test it performance to discover its strong points and drawbacks, to finally, in section 4 draw the final conclusions.

2 SONAtA. A SON Classifier

The SON Classifier function in carried out by SONAtA (SON AuToorganization Algorithm). SONAtA classifies users automatically into semantic groups from a populated ontology. It must be noticed that SONAtA is executed locally in each user, once the user first enters the system (i.e. bootstrapping) and then periodically. Processing locally the information, a global and decentralized organization of the existent users of the P2P network into SONs is achieved naturally. Users must share the same domain ontology, or, at least, a part of it.

The algorithm is based on the idea that some distinguished elements (concepts and individuals of such concepts) may form a SON. Which distinguished elements form a SON depends on the number of individuals related with that concrete distinguished element (we will discuss on this later).

For each distinguished element of a given ontology, the number of individuals from such distinguished element would be the number of items the user has about that content. However, in certain occasions, an individual as such does not contain all information about the user's content. We need to navigate through the ontology (i.e. follow relations) to find that information. If so, what makes sense is to guess how many times that individual has been used. It is also possible that this individual is not used directly, but from indirect relations. On the other hand, not everything related directly or indirectly with an individual makes sense to be used. To solve this, SONAtA only uses relations that denote abstraction.

As we have seen above, a crucial element of its way of working is the idea of relation. Through a relation, a distinguished element is linked to another in the ontology. Sonata will be interested only in those links that denote abstraction. In our system a relation denotes abstraction (e.g. *partOf*) if its domain is more specific than its range, which is more general. These abstraction relations will be useful to detect the referencing level of an item.

An ontology can be viewed as a graph where nodes are the distinguished elements and arcs are the properties. SONAtA will need to walk through the graph. To do this, it will manage two types of relations in terms of distance between the distinguished elements that links to.

Definition 1 (Direct relation between distinguished elements). *Let a and b be two distinguished elements from an ontology O. Then, a is directly related to b if such relation has a as domain and b as range.*

But we are interested as well in relations through more than one property. What the algorithm tries to determine are interests of the user using the content the user owns, and may be insufficient to use just direct ramifications. Hence the following definition.

Definition 2 (Indirect relation between distinguished elements). *Let a and b be two distinguished elements from an ontology O. Then, a is indirectly related to b if there is a path to reach b from a through two or more direct relations.*

There will be particular properties (e.g. subclass relation, such as in OWL) denoting abstraction. And therefore, they could be automatically recognized by SONAtA. However, an ontology modeler can define new properties ontologies that use as abstraction relations, without specifying. It will be required to label as such those properties, so that SONAtA can be applied. A preprocessing of the ontology as a prior step to the application of SONAtA is needed. If an ontology is well labeled, it will have no cycles. This assumption does not detract genericity to the algorithm as it simplifies it, because if any cycle exists, there should be a distinguished element more abstract and concrete at the same time than other. Now, we can define a reference by abstraction.

Definition 3 (Reference by abstraction). *Let a and b be two distinguished elements from an ontology O. Then, a references by abstraction b if a is related directly or indirectly with b.*

Actually, for every distinguished element of the ontology, SONATA is interested in knowing if it is enough important for the user to create/join a SON from it. To this end, we analyze how much a user uses (instances it) a distinguished element or how much content owns about this distinguished element (references to the distinguished element), which corresponds exactly with its extended cardinality. This is calculated from the extended cardinality set.

Definition 4 ECS (Extended Cardinality Set) *of a distinguished element c from the ontology O is defined recursively as the union of the Extended Cardinality Sets of all distinguished elements that directly reference it by abstraction.*

If c has no abstraction references, its ECS is equal to c only if it is an individual. Otherwise, it is equal to the empty set.

Every individual in the ECS can be labeled as *used*, *not used* or *active*. Individuals are labeled as *not used* by default. When a distinguished element becomes an active SON, individuals from its ECS are labeled as *active*. As ECS is calculated as the union of other ECS, if an individual is available in more than one ECS, it is labeled as *not used* if it is labeled as *not used* in any of the ECS, otherwise, is labeled as *used*. From all these considerations we can define the extended cardinality.

Definition 5 (Extended Cardinality of a distinguished element). *Let a be a distinguished element of an ontology O. The extended cardinality of a is calculated with the number of individuals from the ECS of a, and with the number of individuals instancing the concept (if it is a concept), being the higher value its extended cardinality.*

The extended cardinality is useful to know the interest of a concrete user about any distinguished element, since it reflects to what extent it has been used it has been used in the ontology, directly as instantiation or by individual's references. SONAtA classifies the user in semantic groups taking into account mainly the extended cardinality of the user's dintinguished elements. Depending on certain features the peer can assume two roles in a SON. A peer p is **Active Peer** in a SON s if p is actively involved in s: perform searches within s, looking for similar content and users in s, or any operation that requires interaction with other peers. Otherwise, it will be **Passive Peer**, and will only be participate in s by request of other peers. Active peer is the role to assume in the concepts that best define the user based on its content. The other role, passive peer, completes the user profile. The user is interested in joining the SONs more concrete as possible, regarding its profile, since they will achieve a higher affinity with the peers in the SON. However, users of a more general SON will be interested in users of specific SONs, thus are part of the overall by hierarchical inclusion. Therefore, a user from a more concrete SON should be part of the more general SONs passively responding requests from active users of those SONs.

2.1 Algorithm

An algorithm capable of classifying each peer in different SONs depending on their content is proposed here. This classification is done locally and independently at each peer, allowing the peer to be clustered in groups with other semantically similar peers, without having prior knowledge of the network. They just need a common ontology, or at least, a common base ontology, that may be extended. Every distinguished element within the ontology is a potential SON named as that distinguished element (e.g. Rock music SON, 80s SON, ...). Besides, any combination of distinguished elements with enough content in common from the ontology could represent a SON (e.g. Rock music from the 80s SON). There are two types of SONs. **Basic SONs** and **combined SONs**. Basic

SONs correspond to basic individuals or concepts in the ontology. A combination of basic SONs is a SON which holds individual which would be in all the basics SONs composing the combined SON. Thus, a combined SON is the intersection of any combination of basic SONs. We now define when a distinguished element from an ontology become a feasible SON:

Definition 6. *Given a distinguished element c from an input ontology O, already populated, we say that c is valid as a feasible SON if:*

$$(\theta(c) \geq l) \vee \left(\frac{\theta(c)}{t} \geq p\right) \tag{1}$$

where t is the total number of individuals in the ontology O (its cardinality). p is the minimum ratio out of t of distinguished elements needed to join a SON, from 0 to 1. l is the minimum number of distinguished elements needed to join a SON. And $\theta(c)$ (Extended Cardinality) defines the number of valid individuals or references from other individuals for the distinguished element c. The algorithm works as follows:

For every distinguished element in the ontology, the algorithm calculate its ECS (*Algorithm 1*), finding all feasible basic SONs after that, using its extended cardinality as described above. If it occurs that only individuals labeled as *not used* are found in the ECS of the distinguished element, and it is still a feasible SON, that distinguished element is an *Active SON*. Otherwise it is a *Passive SON*. The next step is determining all the combined SONs. For each pair of not related Active SONs (not accessible by abstraction relations), the algorithm checks if there are enough *active* individuals that fit in both SONs at the same time. That is, intersection between the two SONs. If the intersection between

Algorithm 1. ecs

```
 1: Let $c$ be the input concept or individual
 2: $ecs \leftarrow \emptyset$ //$ecs$ will store the Extended Cardinality Set of $c$
 3: $rc \leftarrow referringConcepts(c)$ // returns all concepts that reference $c$ by abstraction
 4: if $rc \equiv \emptyset$ AND $c$ $isIndividual$ then
 5:     $ecs \leftarrow c$
 6:     return $ecs$
 7: end if
 8: for all $e$ in $rc$ do
 9:     $ecsAux \leftarrow ecs(e)$
10:     for all $i$ in $ecsAux$ do
11:         If ($i \notin ecs$) OR ($i$ is $NOT\_USED$)
12:             $ecs \leftarrow ecs + i$
13:     end for
14: end for
15: if $countNotUsed(ecs) > fi$ then
16:         $setActiveToNotUsed(ecs)$ // All individuals set as NOT_USED are changed
    into ACTIVE
17: return $ecs$
```

the SONs has enough individuals to create a new SON (i.e. is a feasible SON as defined before), a SON as a combination of the two SONs is created. This step is done iteratively with the new formed SONs until no new SONs are formed.

Once all the combined SONs are selected, the algorithm must set as *used* all the individuals used to create the combined SONs from the basic SONs to which they are part of. After that, SONs which remain as **active SONs** are also set. For each SON (combined or not), we must check if there is any combined SON that includes that SON (i.e. that SON is actually combined with other). If so, all the individuals common in both SONs are marked as *used* in the simplest SON, because the combined SON is more concrete than the other, thus, that is the one which must remain as *active*. The other SON will still remain as *active* if it has enough not common *active* individuals.

Once the algorithm is presented, the following are the most important ideas behind it. Feasible SONs are SONs that the user is interested on, they can be either active or passive, depending if the individuals have been already used to create other active SONs or not. If there are just a few bad classified individuals, it does not affect the algorithm, since a minimum quantity of individuals are needed to fulfill a SON. There are two main requisites to form a feasible SON, minimum number of individuals and minimum percentage of individuals. Percentage is useful when the peer does not have so many individuals. We just look for the most used. On the other hand, the minimum umber of individuals is used to avoid an overwhelming use of some concept with respect to the rest.

3 Evaluation

In this section the algorithm is evaluated in order to test its performance. We created some initial experiments distributing individuals among different concepts, varying the number of concepts and individuals, up to 2,500 concepts and 40,000 individuals, with no abstraction relation between distinguished elements. Execution time was quite low (below one second) and linear with respect to the sum of concepts and individuals. If there are no relations of abstraction, the algorithm has no charge. Next step was testing how abstraction relations affect the algorithm execution. We created the following experiments, each one with different ways of relating individuals through abstraction references. They all have in common the disposition of concepts. Each concept is referenced by a more specific concept than it, except the most concrete, which is not referenced by any. These are the different experiments:

1. **C1. Random distribution of individuals, references only to next distinguished element (***Figure 1 (a)***).** In this experiment individuals are distributed among concepts randomly by using a uniform distribution probability. Each individual only references one individual, randomly chosen from all individuals from the immediately following concept more abstract than the concept it belongs to. The average reference length will remain low. This experiment aims to observe how the algorithm behaves when the distinguished elements of ontology are weakly linked (not everything is related to everything).

2. **C2. Individual distribution on the edge, references only to the next distinguished element (*Figure 1 (b)*).** In this experiment, each concept is instantiated only once, except for the first concept (the most concrete), where are all other individuals. All individuals in that concept reference by abstraction the only from the concept that is referenced by the instantiated concept. The purpose of this test is evaluating how the algorithm behaves with a case fairly common in an ontology: the most specific concept instances most of the individuals (i.e. the rest concepts instance a single individual, as shown in Figure). In turn, all individuals of the most concrete concept are related with the individual instanced by the immediately more abstract concept. This, in turn, is related also with the next more abstract, until reaching the superconcept of all of them.
3. **C3. Individual distribution on the edge, total reference between distinguished elements (*Figure 1 (c)*).** The difference between this experiment and the previous one is that in this one, each concept and individual references, not only the following more abstract concept, but all concepts and individuals more abstract directly. It corresponds with the maximum referentiation level, with a number of references equal to $T_n = \frac{n(n+1)}{2}$, where n is the number of concepts. With this experiment we aim to discover how does the fact of performing the union of all ECS of distinguished elements referencing affects the algorithm, and the empirical time complexity evolution it offers when comparing with the previous ones.

Results can be seen in Figure 1 (d, e, f), corresponding experiments C1, C2 and C3 respectively. Time is expressed in seconds, varying the number of concepts and instances. From all of them, the one requiring less processing time is C1, C3 requires the most time, with C2 as a medium case. In C1, we can observe an almost linear behavior with respect to input, with slight variations due to the random nature of the test. This is because not all individuals will walk through all concepts, but only a part of them depending on where they are located and who they reference to. As we can see in C2, there is a slightly exponential growth respect the number

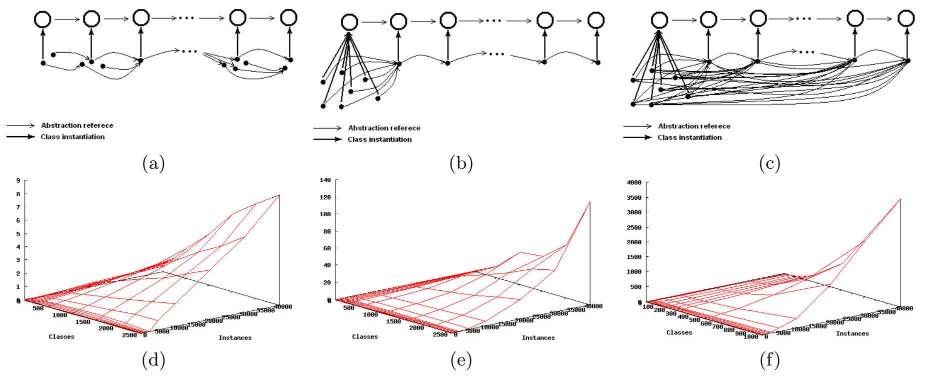

Fig. 1. Experiment structures and results

of concepts and individuals. This is mainly because all individuals from the most concrete concept have to reach the most abstract concept, following references, crossing each concept, due to the shape of the ontology. In C3, since all are referencing each other, we can see how time complexity grows not linearly but a higher rate. In addition to move individuals through all concepts, since all are referenced, all of them must be processed in each distinguished element of the ontology, calculating the union of all the distinguished elements which reference it, requiring more memory and processing time for execution.

From this data we can highlight that what really contributes to increase execution time of the algorithm are two parameters. On the one hand, the length of references by abstraction. When a distinguished element of the ontology is referenced, the algorithm should check all distinguished elements from the ECS of each distinguished element referencing it. On the other hand we have the referencing index. The more referenced a distinguished element is, the more ECS must combine to create its own ECS.

4 Conclusions

In this paper, we have presented SONAtA, an algorithm to organize peers into semantic groups from an ontology. Even though it has been designed to classify peers into Semantic Overlay Networks, SONAtA can be used for other tasks such as automatic clustering of users semantically similar. The only requisite needed to use SONAtA is the existence of an ontology describing the contents from the users, shared, at least partially, by the users in the system we want to cluster. We have evaluated SONAtA, showing that results are acceptable within conventional and even large ontologies as input.

References

1. Crespo, A., Garcia-Molina, H.: Routing indices for peer-to-peer systems. In: 22nd International Conference on Distributed Computing Systems, pp. 23–32 (2002)
2. Garcıa-Sola, A., Botıa, J.A.: Semantic Overlay Networks for Social Recommendation in P2P. In: International Symposium on Distributed Computing and Artificial Intelligence 2008 (DCAI 2008), pp. 274–283 (2008)
3. Crespo, A., Garcia-Molina, H.: Semantic Overlay Networks for P2P Systems. Springer, Heidelberg (2004) (submitted for publication)
4. Tang, C., Xu, Z.: Peer-to-peer information retrieval using self-organizing semantic overlay networks, pp. 175–186. ACM Press, New York (2003)
5. Aberer, K., Cudre-Mauroux, P., Hauswirth, M.: GridVine: Building Internet-Scale Semantic Overlay Networks, pp. 107–121. Springer, Heidelberg (2004)
6. Cudre-Mauroux, P., Agarwal, S., Aberer, K.: GridVine: An Infrastructure for Peer Information Management 11, 36–44 (2007)
7. Loser, A., Staab, S.: Semantic Social Overlay Networks. IEEE Institute of electrical and electronics 25, 1 (2007)