Multi-Layer Clusters in Ad-hoc Networks - An Approach to Service Discovery

Michael Klein and Birgitta König-Ries

Institute for Program Structures and Data Organization
Universität Karlsruhe
D-76128 Karlsruhe, Germany
{kleinm,koenig}@ipd.uni-karlsruhe.de
http://www.ipd.uni-karlsruhe.de/DIANE

Abstract. One of the core functionalities needed in ad-hoc peer-to-peer networks is service discovery. However, none of the existing solutions for service discovery work well in these dynamic, decentralized environments. Therefore, in this paper, we propose a new approach to service discovery which is based on the dynamic organization of the services into multi-layer clusters. These clusters are formed based on both physical and semantic proximity.

1 Introduction

Anna is a graduate student majoring in computer science. Currently, she prepares for the finals in her information systems class. She has downloaded some of the official course material onto her laptop computer. She also stores the solutions to some exercises and her personal summary of the first few chapters there. Of course, Anna is not the only student about to take the exam, others are in the same situation. They, too, will have stored some information on their computers - information that might be a good complement to the one Anna already possesses. Thus, the idea to connect the computers of these students to a peer-to-peer network and to allow the exchange of information stored on these devices is pretty natural. However, a classical peer-to-peer network (if such a thing exists) is not sufficient for our needs. Like Anna, some of the students will own laptop computers or other mobile devices and will want to have access to the network from wherever they are at a certain point of time regardless of the existence of an underlying network infrastructure. Therefore, in the DIANE project [1], we attempt to use wireless ad-hoc networks to offer the functionality described.

Assuming we have succeeded in technically connecting the computers to such a network, what is needed next is a possibility for the students (or their computers) to find information (or more generally services) in the network. Due to the distributed and dynamic nature of ad-hoc networks, a centralized approach which is based on a service directory is not feasible. Because of the characteristics of wireless networks, a Gnutella-like gossiping approach involving a large

message overhead is not feasible, either. Therefore, we present a new approach to discovering services in wireless ad-hoc peer-to-peer networks.

The basic idea of our approach is to take advantage of the fact that nodes in an ad-hoc network are naturally grouped along two dimensions: The first of these dimensions is the physical proximity of nodes; the second is their semantic proximity. Two nodes are considered to be physically close, if they are within radio range of each other. Two nodes are considered to be semantically close, if the services they offer are similar. For our work we assume that a common ontology exists within the network that is used to describe the services offered by a node. The structure of this ontology is very simple: Terms are related via isSubTopicOf links only. Thus, if two nodes offer services described by the same term or by terms that are close in the ontology, they are considered to be semantically close.

Based on these dimensions we now define multi-layered clusters on the nodes which will be used for service discovery. Let us first have a look at how these clusters are defined. Afterwards, we are going to explain how the clusters can be used to find specific services. The bottom layer of our cluster hierarchy, the leaf level, is formed by grouping nodes that offer services that are described by the same leaf term of the ontology and that form a connected sub net. On the next layer several of these leaf-level clusters are combined to form larger clusters offering services that belong to the same term on the next level of the ontology and so on.

This cluster hierarchy can be used for service discovery as follows: When a node looks for a certain service, it first checks, whether that service is available in its own leaf-level cluster. If this is not the case, the query is moved to higher-level clusters until one that might contain an appropriate service is reached. In this cluster, the query is moved back down through the hierarchy to the appropriate leaf-level cluster. In the following sections, we will describe the details of this process.

The remainder of this paper is organized as follows: In Section 2 we review related work. Then, Section 3 describes multi-layer clusters in more detail. Section 4 explains how this concept is used for service discovery. The paper concludes with a discussion of the advantages of our approach and an outlook to future work (Section 5).

2 Related Work

This section contains a short introduction to ad-hoc networks in general and an overview of approaches to service discovery taken by different research communities.

An ad-hoc network, as the name suggests, is a spontaneous community of mobile devices communicating via wireless links without the support of an underlying infrastructure [3]. In the simplest case, all the devices are within radio-range of each other, thus forming a single-hop ad-hoc network. More interesting are multi-hop ad-hoc networks, where devices may need the help of interme-

diate nodes to communicate with one another. In this case, the intermediate nodes take over the functionality classically provided by routers. In an ad-hoc network, nodes interact as equals, thus, they can be regarded as a special kind of peer-to-peer network. However, in contrast to peer-to-peer networks in wired environments, a number of additional problems need to be solved in ad-hoc networks. The most obvious of these problems is the need to offer a possibility to route messages in ad-hoc networks. The topology of an ad-hoc network may change frequently. The nodes that take part in an ad-hoc network typically have a limited power supply only, thus restricting the amount of messages they can process and in particular send. These characteristics make the development of appropriate routing protocols a challenging task. Thus, in the past, most of the research effort has been concentrated on this topic. Much of this work has been done in the context of the IETF MANET working group [4].

While routing protocols determine a way from a given node A to a given node B, service discovery protocols try to find a node B that is able to fulfill a certain task that node A is looking for. Traditionally this problem is solved by maintaining a directory server that contains the necessary information to identify node B. A typical representative of this class of solutions is Jini [5], a Java based network technology that aims at facilitating integrated usage of services in dynamic networks. The core component of a Jini system is a service directory, the Lookup Server. Devices that offer services register with this directory. Devices looking for services query the directory and obtain an appropriate service stub from the directory. Other approaches like UPnP [6], the Corba Naming and Trading Services [7], the Bluetooth Service Discovery Protocol [8], operate similarly. The same is true for Napster [9] and related systems.

For ad-hoc networks, approaches that rely on a directory server do not apply for two reasons: First, nodes can join and leave the network at any time, thus the set of services offered in the network changes frequently, making it difficult to maintain an up-to-date directory. More severely, in an ad-hoc network, there is no guarantee, that a certain node will be available at any given time. Thus, in a true ad-hoc network, none of the participating nodes could serve as a directory server.

Similar problems have resulted in peer-to-peer systems trying to avoid the need for a centralized service directory altogether. Gnutella [10], for instance, relies on a gossiping approach that propagates searches throughout the network. Its main drawback is that they necessarily involve a large message overhead. This may be acceptable in wired networks; however, it is a major disadvantage in wireless environments with limited bandwidth and limited power resources, where minimizing communication is a major requirement.

A major prerequisite for service discovery is that the devices have agreed on some common scheme to describe services. Jini, for example, uses the Java class hierarchy to achieve this goal. In our work, we assume that services are described by a common ontology [2] that is known to every node.

To summarize, in order to support service discovery in a mobile ad-hoc network, we need an approach with the following characteristics: It allows to find

services without relying on a centralized directory server. At the same time, it minimizes message overhead and is able to deal with the high dynamics and ever changing topology of ad-hoc networks. In the next sections, we present such an approach.

3 Ad-hoc Nets as Multi-Layer Clusters

In this section, we describe how ad-hoc networks can be conceptually organized into multi-layer clusters. After introducing this structure, we will explain in Section 4 how it can be used to support efficient service discovery.

The main idea of our approach is to naturally combine the conceptual structure given by an ontology with the technical structure of the devices caused by the current state of reachability between the nodes. We present the details of this process in this section starting with explaining and formalizing the two different types of structures that have to be merged: the ontological structure and the technical structure. After that, we combine the two structures to form multi-layer clusters.

3.1 The Ontological Structure

The network consists of devices that offer services. In the context of this paper, we are interested in information services, i.e. services that deliver documents. To describe the content of information that is delivered by information services, we use very simple hierarchical ontologies consisting only of a set of terms $T = \{t^1, \ldots, t^r\}$ and the relationship type *isSubTopicOf*. Figure 1 shows a sample ontology for the database domain. The top most term (here `database`) is called root term and comprises the whole domain of the ontology. The other terms are subordinated to this root term. We use the notation $t^1.isSubTopicOf = t^2$ to denote the relationship between terms t^1 and t^2.

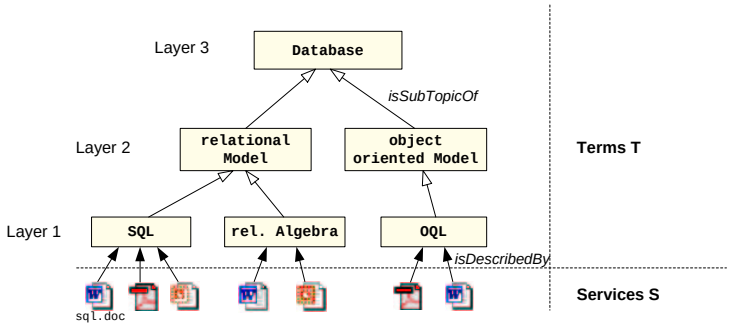

Fig. 1. A database ontology and the documents described by it.

For ease of presentation, we make the following assumptions throughout the remainder of this paper:[1]

- Each service offers exactly one document.
- Each document s can be described by exactly one leaf term t of the ontology: $s.isDescribedBy = t$.

On the ontology and the set of documents described by it, we can define a *parent function* as follows:

Definition 1 (Parent Function).
Let $S = \{s_1, \ldots, s_n\}$ be the set of all services and let $T = \{t^1, \ldots, t^r\}$ be the set of all terms in the ontology with root term t^r. Then, we define the parent function $p : S \cup T \to T$ *as:*

$$p(x) = \begin{cases} t, \text{ with } t = x.isDescribedBy & : \quad x \in S \\ t, \text{ with } t = x.isSubTopicOf & : \quad x \in T \backslash \{t^r\} \\ x & : \quad x = t^r \end{cases}$$

We will use $p^k(x)$ as a shorthand notation for $\underbrace{p(p(\ldots p(}_{k-times} x) \ldots))$ in the remainder of the paper.

As the ontology should form a tree, we explicitly prohibit cycles:

$$\forall t \in T \;\; \forall k \geqslant 1 : p^k(t) \neq t \;\vee\; t = t^r.$$

In our example, we have $p(\texttt{sql.doc}) = \texttt{SQL}$ and $p^2(\texttt{OQL}) = \texttt{database} = t^r$.

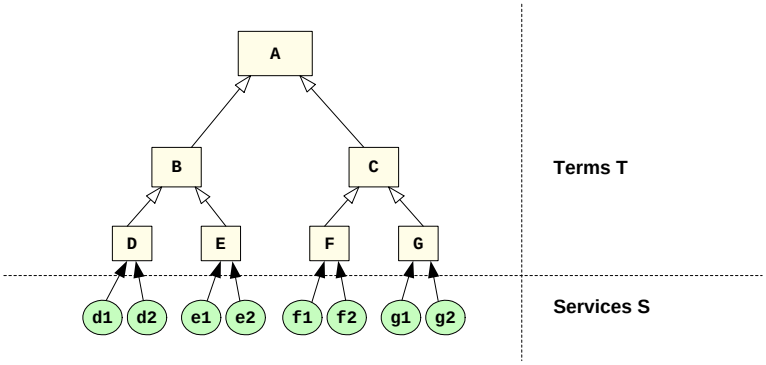

Fig. 2. Generic ontology, which serves as example ontology for the remainder of the paper.

[1] Note that the approach will also work if we relax these assumptions. However, the resulting presentation would be a lot less elegant.

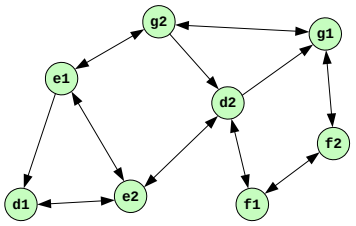

Fig. 3. Typical scenario of radio connectivity between devices. Because of different power consumption for sending and receiving packets, the edges can be directed.

The number of layers n in an ontology is the number of nodes on the longest path from a leaf to the root term. Whenever the ontological layer to which a term t belongs is important, we will write t_l with l being the layer, i.e., n minus the distance to the root.

For our example ontology in Figure 1, n is 3, the layer of the term `OQL` is 1, the layer of the term `Database` is 3. For the remainder of the paper, we will refer to the generic ontology and services in Figure 2.

3.2 The Physical Structure

Besides the ontological structure, we have to take into account the physical topology of the devices. For our paper, we make the following assumption:

- Each device in our ad-hoc network offers exactly one service $s \in S$. Devices offering no services are useful only for routing purposes and are not further considered in this paper. Devices offering more than one service can be regarded as several devices with just one service respectively, standing in radio contact to each other all the time (see reachability below). If a document is offered on more than one device, each of these devices offers its own service instance of the document. Thus, devices are uniquely identified by the service they offer. We will therefore denote devices with the service variables $s_1, s_2, \ldots s_n$.

Devices are connected directly to all the devices within their radio range. Based on these connections, reachability can be defined as follows:

Definition 2 (Device Reachability).
The dynamical relation $\dashrightarrow \subseteq S \times S$ *describes the* reachability *between the devices in the ad-hoc network.*

Single-hop reachability $i \dashrightarrow j$ *is given iff. device* $i \in S$ *is currently able to send information to device* j *via a single-hop connection.*

As this reachability is transitive, we have multi-hop reachability $i \overset{*}{\dashrightarrow} j$ *iff.* $i \dashrightarrow \ldots \dashrightarrow j$.

Figure 3 shows a typical scenario of devices and their current reachability. E.g., we have `e1` $\dashrightarrow$ `d1` and `f2` $\overset{*}{\dashrightarrow}$ `e2`.

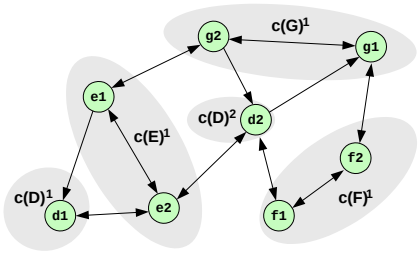

Fig. 4. Clustering on the leaf level. Devices are grouped together if they belong to the same term and reach each other.

3.3 Combination of the Structures: Multi-layer Clustering

Up to now, we have looked at the two types of structures separately. However, in a real net, the two cannot be isolated from each other. In this section, we describe how devices can be organized into clusters so that each cluster contains devices that are both physically close to each other and offer services that are semantically close. On the bottom layer, we form clusters of devices that offer services described by the same leaf term of the ontology and that are within radio range of each other. On the higher layers, we combine these clusters to clusters whose semantic description becomes more general when moving up the ontology tree. On the top layer, this results in one big cluster containing all the services offered within the net. (If the network is partitioned, we will have more than one cluster at this layer, too.)

In Figure 4, the clustering on Level 1 (i.e. the leaf level) is performed on the net of Figure 3 and with the help of our generic ontology. As a rule, a set of devices forms a cluster on Level 1 iff. (a) all offered services are described by the same leaf concept and (b) every two devices from this set are mutually reachable.

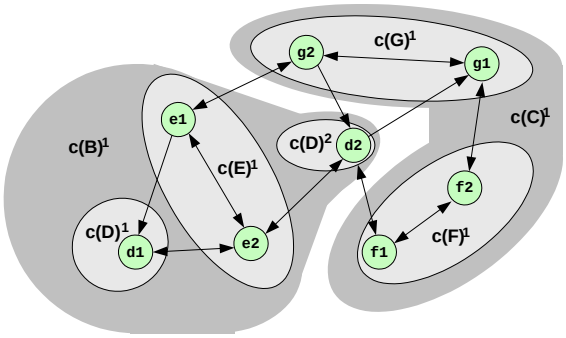

Fig. 5. Cluster are grouped to larger clusters according to higher levels of the ontology and the connectivity between clusters.

In the example, five rather small clusters have been formed by this definition. Note that this cluster construction rule is non-ambiguous as no parameters like cluster size and shape need to be determined. Only varying radio connections and varying service offerings can change the clustering topology.

On the next layer, we combine clusters that offer services described by the same next layer term in the ontology and that are mutually reachable from a cluster point of view (see below). Figure 5 shows how this is done with the clusters of Figure 4. On the left side, the services offered by clusters $c(D)^1$, $c(E)^1$ and $c(D)^2$ have the same parent term B and are mutually reachable, so they are grouped to the new cluster $c(B)^1$. On the right side, new cluster $c(C)^1$ is formed. The last clustering step, which is not depicted here, would join those two clusters to one large cluster $c(A)^1$.

Let us now give a formal definition of a cluster on layer l:

Definition 3 (Cluster).
A cluster $c_l(t_l^m)$ *on layer* l *with respect to term* t_l^m *on ontology layer* l *is a set of clusters or devices on layer* $l-1$*, such that*

$\underline{\text{for } l=1:}\ c_1(t_1^m) \subseteq S$ *with*
a) $\forall s_i \in c_1(t_1^m) : p(s_i) = t_1^m \ \ \wedge$
b) $\forall s_i, s_j \in c_1(t_1^m) : s_i \overset{*}{\dashrightarrow} s_j$

$\underline{\text{for } l>1:}\ c_l(t_l^m) \subseteq \{c_{l-1}(t_{l-1}^1)^1, \ldots, c_{l-1}(t_{l-1}^1)^{m_1}, c_{l-1}(t_{l-1}^2)^1 \ldots, c_{l-1}(t_{l-1}^n)^{m_n}\}$ *with*
a) $\forall c = c_{l-1}(t_{l-1}^i) \in c_l(t_l^m) : p(t_{l-1}^i) = t_l^m \ \ \wedge$
b) $\forall c, d \in c_l(t_l^m) : c \overset{*}{\dashrightarrow} d$

Here, t_l^i *stands for the* i*-th term on ontology level* l *and* $c_l(t_l^i)^j$ *is the* j*-th cluster on level* l *for term* t_l^i*.*

Informally, a cluster on layer l groups together clusters/devices of the next lower layer that belong to the same parent term and are mutually reachable. The definition above uses cluster reachability, which we haven't introduced yet. Figure 6 shows intuitively, under which conditions clusters are reachable: Cluster $c(E)^1$ can reach cluster $c(G)^1$ because they can communicate via their directly connected devices `e1` and `g2`. Generally speaking, one cluster c reaches another cluster d if c contains a device/cluster that is directly connected to a device/cluster in d.

Before we formally define cluster reachability, we define a layer of the cluster hierarchy as the set of all clusters on that layer:

Definition 4 (Layer l of the Cluster Hierarchy).
A layer l of the cluster hierarchy *is defined as follows:*

$$C_l := \{c_l(t_l^1)^1, \ldots, c_l(t_l^1)^{m_1}, c_l(t_l^2)^1, \ldots, c_l(t_l^k)^{m_k}\}$$

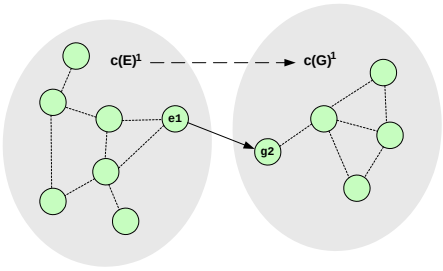

Fig. 6. Intuitive cluster reachability. Cluster $c(E)^1$ reaches cluster $c(G)^1$ because it can send messages over the directly connected devices `e1` and `g2`.

Additionally, we define C as the set of all clusters regardless of their layer:

Definition 5 (Set of all clusters).
The set C of all clusters *defined as follows:*

$$C := C_1 \cup C_2 \cup \ldots \cup C_n$$

Now, we have all the prerequisites needed to formally define cluster reachability. For that, we extend the $\dashrightarrow$ relation from above to clusters:

Definition 6 (Cluster Reachability).
$\dashrightarrow \subseteq C \times C \ \cup \ S \times S$. *Let* c *and* d *be two clusters from* C. *We have*

$$c \dashrightarrow d \ \ \textit{iff.} \ \ \exists i \in c \ \ \exists j \in d : \ i \dashrightarrow j$$

In this case, we mark i and j as so called *switching devices/clusters*, which will help us later to send a message from cluster c to cluster d. In our example in Figure 4, the devices `g1` and `f2` act as switching instances for the two clusters $c(G)^1$ and $c(F)^1$.

Now that we have introduced the cluster hierarchy, we can define a parent function on clusters. In analogy to the parent function for terms, this function determines which more general cluster a cluster is part of. Thus, while the parent function for terms operates along the semantic dimension, the parent function for clusters operates along the topological dimension:

Definition 7 (Parent Function of a Cluster).
We define $\bar{p} : S \cup C \rightarrow C$, *which assigns the parent cluster to services as well as clusters:*

$$\bar{p}(x) := c_i(t_i), \ \ \textit{if} \ x \in c_i(t_i)$$

4 Service Discovery via Multi-Layer Clusters

In this section, we describe how the multi-layer clusters introduced in the last section can be used for service discovery.

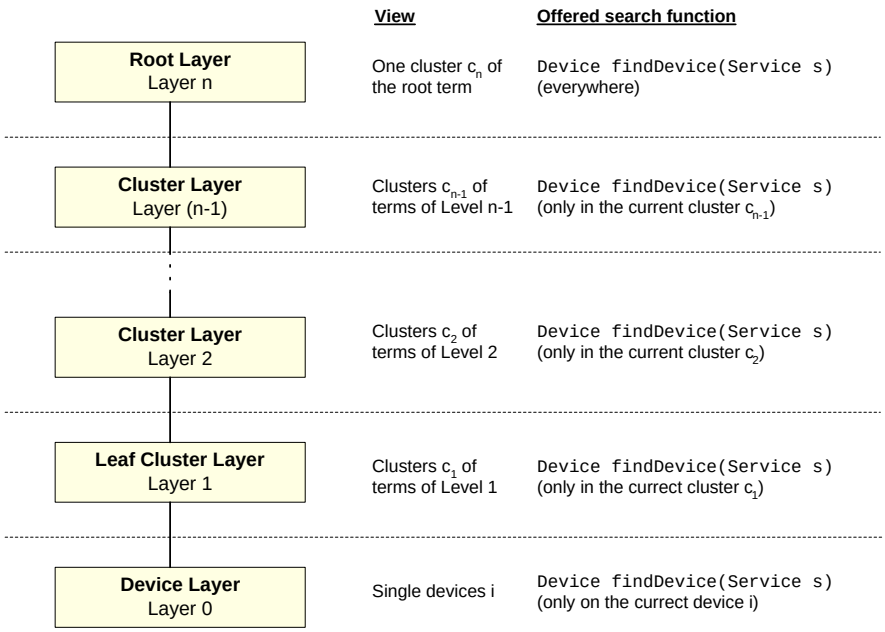

Fig. 7. The layer architecture. The search functionality as well as the cluster sizes increase with every network layer, the number of clusters decreases.

4.1 Layer Architecture

The clustering we described in the previous section leads to the layer architecture depicted in Figure 7. As in every layered architecture, the task of providing complex functionality (here to find a device offering a given information service somewhere in the net) is broken down into several steps each of which increases the functionality by using the methods of the underlying layer. Our architecture consists of $n + 1$ layers, one for each of the n ontology layers and one for the lowest device layer. Each layer has a different view on the network and offers two methods: a search and a send method. The first method checks whether the service is available in the current cluster or device; the second one is used to send a message to another cluster or device. Both methods operate within one layer only and are implemented using the respective functions of the layer below.

The bottom-most layer of our architecture is the **device layer** (or Layer 0), where the network is seen as a collection of single devices connected according to the basic reachability relation $\dashrightarrow$. Each device i offers one very simple search function that checks whether a given information service request can be met on the own device. If so, it returns i, else it returns `NULL`:

```
Device findDevice(Service s)
```

For this very basic layer no ontology is needed - the device answers the request just by checking its own service description. As `findDevice` will not always be

successful on the current device, the request has to be redirected to other devices. Thus, to send a message (for instance a find request) to a neighboring node with a single hop, we can use the function

```
void sendTo(Device j, Message m)
```

This function results in the message being sent if $i \dashrightarrow \texttt{j}$ and will return an exception else.

Above the device layer, we find the **leaf cluster layer** (or Layer 1). Here, the network is regarded as a collection of clusters consisting of services described by the same leaf terms of the ontology. A cluster $c_1(t_1^j)$ offers the following search function

```
Device findDevice(Service s)
```

with $p^1(\texttt{s}) = t_1^j$. It searches for a device in the cluster c that offers service `s`. As clusters are connected logically via switching devices, we have the following sending function between two neighboring clusters:

```
void sendTo(Cluster d, Message m)
```

which will result in the message being sent if $c \dashrightarrow d$, and an exception otherwise.

Analogously, we have the functions `Device findDevice(Service s)` and `void sendTo(Cluster` d`, Message m)` on all the higher **cluster layers** (Layer 2 to Layer $n-1$). With each layer, the service descriptions we are looking in becomes more general and the physical area we are searching becomes bigger. Finally, the topmost **root layer** (or Layer n) offers the desired function

```
Device findDevice(Service s)
```

without any constraint to `s` because $p^n(\texttt{s}) = t^r$ for all $\texttt{s} \in S$.

4.2 Implementation of Layer Functions

After having introduced the declaration of the available functions in each network layer, we want to show how to implement them. We can distinguish two general approaches: First, an omniscient implementation, which does not take into consideration that the nodes are physically distributed and method invocation messages have to be sent. Such an implementation cannot be used as a basis for a network protocol and therefore is not suitable for our purposes. Thus, we will present an implementation according to the second approach, that is a local implementation, which only considers the locally available information and needs to send method invocation methods to other nodes. The transformation of such an implementation into a network protocol is rather straightforward.

On the device layer, the two functions can be implemented very easily. A device i, on which `Device findDevice(Service s)` has been called, just compares the service description it stores and returns i if it matches, and `NULL` otherwise. The `void sendTo(Device j, Message m)` is simply delegated to the lower MAC layer if $i \dashrightarrow j$ holds and returns an exception otherwise.

The functions in the Layers 1 to n are a bit more complicated. Generally, if the function `findService` is called in cluster c_l from device i, the `findService` function of all subclusters $c_{l-1}^1, \ldots c_{l-1}^m$ is called. If one or more of these functions deliver a concrete device, one of those is chosen and returned, otherwise `NULL` is returned. It is important that the concrete execution depends primarily on the order in which the subclusters are visited. The pseudocode of the algorithm can be found in Figure 8. It is written for a call in cluster c_l from device i, where c_l is a cluster on Layer $l \geq 1$. First of all, in line `(02)`, the currently visited cluster is marked. We determine the current subcluster c_{l-1}, i.e. the subcluster that contains device i in line `(03)` and its ontological term t_{l-1} in line `(04)`. If this term matches the type of the requested service `s` and if the cluster is still unmarked (line `(05)`), we start a new search in it in line `(07)`. If a concrete device is found, this is returned to the caller in line `(08)`. Up to now, the search has been strictly local. We have only descended in lower layers of the part of the network the query originated from. Only if this search is not successful, the algorithm will now gradually extend the physical area in which it searches by looking at neighboring clusters. This is done by sending the `findService` query to the next subcluster of the current cluster c_l. To do that, all possible candidates are determined in line `(12)`. A candidate has to be a cluster or a device that is

```
Processing device i on cluster c_l

(00)     Device findService(Service s)
(01)     {
(02)           mark(c_l);
(03)           c_{l-1} = p̄^{l-1}(i);
(04)           t_{l-1} = p^{l-1}(i);
(05)           if ((p^{l-1}(s) == t_{l-1}) and (c_{l-1} is unmarked))
(06)           {
(07)                 Device result = (c_{l-1}).findService(s);
(08)                 if (result ≠ NULL) return result;
(09)           }
(10)
(11)           //Send query to the next subcluster
(12)           D := {d_{l-1} ∈ c_l | c_{l-1} --→ d_{l-1} }
(14)
(15)           if (D is empty) return NULL;
(16)           repeat q times
(17)           {
(18)                 d := pickFrom(D);
(19)                 sendTo(d, "findService(s)");
(20)           }
(21)     }
```

Fig. 8. Pseudocode implementation of `findService` for a call in cluster c_l from device i. Here, c_l is a cluster on Layer $l \geq 1$.

a descendant of c_l, and can directly be reached by c_{l-1} with one hop. If this set is empty, the search has terminated without success and `NULL` is returned in line `(15)`. Otherwise, one or more of these clusters/devices are chosen by the loop in line `(18)`, to which the `findService` query is sent (line `(19)`).

This algorithms offers some degrees of freedom: First of all, in line `(18)`, the method for fetching an element from the set D is left open. Moreover, the number q of elements is not defined in the condition of the `repeat`-loop. Depending on the knowledge about the network and the characteristics of the network, one of the following exploration strategies can be chosen:

- **Flooding.** The `findService` message is sent to *all* members of D, i.e. $q = |D|$. This method is useful if no further information is available. Its performance is poor as many subclusters are reached more than once and possibly an exhaustive search is executed. Nevertheless, if the topology of the subcluster changes very rapidly, flooding may be the only applicable technique.
- **Cycling.** All subclusters are connected to a ring, thus each member knows its ring predecessor and successor. In this case, we choose d as the successor and send the `findService` message to this cluster only. As in flooding, this can result in an exhaustive search, if, for instance, the suitable cluster is the predecessor of c_{l-1}, but each subcluster is exactly reached once. We can choose this technique, if the dynamics of the subclusters is medium so that a ring structure can be set up. This is not too hard as the structural integrity of a ring can be maintained locally.
- **Direct Routing.** If we have a routing table with an entry for the subcluster d, we send the `findService` message exactly to the next hop denoted in this entry. To speed up the routing, we could possibly add a flag to the message that prevents descending in intermediate subclusters. This technique can only be used if the subcluster movement is very small so that a routing table can be set up.

The function `void sendTo(Cluster` d`, Message m)` is implemented very similar to `findService`. Because each switching point stores information about its partner switching point and the cluster it resides in, `sendTo` can be seen as a search for an appropriate switching point. Analogously, the sending process performs in several network layers and the same exploration techniques like flooding, cycling, and direct routing can be used depending on the current knowledge.

5 Conclusions and further research

In this paper, we presented a new approach to service discovery in ad-hoc networks based on a multi-layered clustering that results from a natural combination of semantic issues (taken from an ontology) and structural issues (based on radio connectivity). These clusters form the basis for a layered network architecture, in which each layer offers a certain functionality basing on the previous layer. The topmost layer allows a user to search for a device offering a given service.

5.1 Advantages of the Approach

Our layer architecture offers several advantages:

Naturalness. The definition of our multi-layer clustering is very intuitive, natural and easy because only two external parameters are taken into consideration: the semantics within a domain ontology and the structure as a graph of radio connections. No additional parameters like cluster sizes or cluster forms have to be tuned as they evolve automatically. In contrast, many other routing or searching protocols require a set of parameters be adjusted accurately for gaining acceptable performance.

Decentralization. The approach does not rely on a central device or an infrastructural component as every device is treated in the same way.

Resource-Awareness. Particularly in mobile ad-hoc networks and their limited resources like battery power and bandwidth, it is crucial that services are used in a resource-aware manner. Thus, a client should use a service that matches its query and is physically close to that client in order to avoid expensive package transmission over several intermediate nodes. Our approach achieves this automatically (i.e. without a cost function) by searching local clusters before accessing physically distant regions. Doing this, we take advantage of the fact that semantically coherent devices are not necessarily physically close. Thus, in principle, a suitable service can be found in every region.

Adaptability. Often, network protocols suffer from the fact that they are specially designed for a certain degree of network stability. In typical ad-hoc networks however, we can spot various regions with highly different dynamics in topology. If we think of a street with two driving lanes leading in opposite directions, for instance, we have a relatively stable network between devices in cars driving in the same direction whereas the network between the two lanes is highly fluctuating. Our clusters are able to adapt to such varying dynamics by choosing an appropriate exploration strategy for each cluster (on each layer) according to the amount of information collected over the time.

Fault Tolerance. In mobile ad-hoc networks, network connections between two devices are not reliable at all: the devices can move out of radio range, they can switch off or fail, and obstacles can disturb the transmission. Many protocols face these problems by setting up a huge list of fault cases in order to handle them. Our approach solves this problem more naturally by moving through the cluster layers and applying exploration techniques dynamically. If, for instance, a connection fails in a cluster on Layer l, the exploration structure in this cluster has to be adapted (for example, routing tables or ring successors will be changed) or changed completely (for example, from cycling to flooding) in more severe cases. Nevertheless, the caller on Layer $l+1$ just continues trying to send its message while now using another way of routing. This technique becomes possible because every node only guarantees to send a message or to find a service, but does not state *how* this is done.

5.2 Future Work

The presented approach leaves some important questions for further research. One main question is the management of administrative data like routing tables, information about ring predecessors and successors, information about switching points, service descriptions, and so on. Within a single instance, these are handled easily by the device itself, but within clusters, we have to decide where to deal with this information. In principle, there are two possibilities: First, electing a cluster head in each cluster and letting this head handle all related tasks. However, this raises the problem, that for clusters on higher levels we approximate a centralized protocol, which is not very robust, if this cluster head fails and not very performant either, as all requests are guided through this head. Another idea could be, to replicate the information on all devices in the cluster so that each device can handle function calls for the cluster on its own. Here, a mechanism needs to be developed that ensures efficient distribution of information even in large clusters. An attractive possibility is a lazy approach, where information is distributed when needed only.

Another important issue that we didn't deal with in this paper is the performance of the approach. Currently, we are implementing a simulation using the QualNet tool [11] and are also carrying out a theoretical analysis of the performance.

Finally, the goal of the DIANE project as a whole is to develop a prototype that will support e-learning. Thus, in the not so far future, on our campus, students like Anna will be able to exchange information as described in the introduction.

References

1. DIANE Project. http://www.ipd.uni-karlsruhe.de/DIANE/en.
2. Birgitta König-Ries, Michael Klein. Information Services to Support E-Learning in Ad-hoc Networks. In: Proc. of 1. Intl. Workshop on Wireless Information Systems (WIS 2002), Ciudad Real, Spain, April 2002.
3. Charles Perkins (ed.). Ad Hoc Networking. Addison-Wesley Publishing Company, 2000.
4. IETF Working Group on Mobile Ad Hoc Networks (MANET). http://www.manet.org.
5. Jini Network Technology. Sun Microsystems. http://www.jini.org
6. Universal Plug-and-Play (UPnP) Forum. Microsoft Corporation. http://www.upnp.org
7. Object Management Group. Trading Object Service Specification. http://www.omg.org/technology/documents/formal/trading_object_service.htm
8. Bluetooth Specification Part E. Service Discovery Protocol (SDP). http://www.bluetooth.com
9. Napster File Sharing. http://www.napster.com, Napster Protocol Specification. http://opennap.sourceforge.net/napster.txt
10. Gnutella File Sharing. http://gnutella.wego.com
11. QualNet by Scalable Networks Technologies. http://www.qualnet.com.