Peer-to-Peer Network for Flexible Service Sharing and Discovery

Andrii Zhygmanovskyi and Norihiko Yoshida

Department of Computer Science, Saitama University, Japan
{andrew,yoshida}@ss.ics.saitama-u.ac.jp

Abstract. In this paper we present P2P-based approach for storing, sharing and discovering services, which can be utilized as a base for agent cooperation in distributed multiagent system. The approach is loosely based on Peer-to-Peer based Web Service Discovery (PWSD) architecture and can be seen as its extension and enhancement. Unlike most other P2P-based approaches it allows flexible search queries since all of them are executed against internal database present at each overlay node. The infrastructure proposed in the paper can serve as a base for wide range of applications including distributed scheduling scenarios and service mash-ups.

Keywords: peer-to-peer, agent cooperation protocol, service discovery, service sharing.

1 Introduction

Nowadays computing is becoming more and more dominated by distributed applications which, in turn, grow to be more complex and require sophisticated protocols for communication and cooperation between participants. On the other hand, many distributed application scenarios can be executed by a set of intelligent agents thus minimizing human participation or eliminating it at all. Since one of the main characteristics of multiagent systems is cooperation, there have already been proposed numerous agent cooperation protocols, such as Contract Net [1] or Recruiting Interaction Protocol [2], which describe an interaction between agents toward finding the most suitable agent for performing given task. However, classical approaches for agent cooperation are foremost designed from the viewpoint of agent capabilities and rarely take the distributed nature of multiagent systems into proper consideration. More specifically, some cooperation models such as teamwork model [3] or joint-intentions model [4] assume there are some shared memory locations that agents can use to share their mental state and goals, but this is difficult to assume for distributed environments. Other models like the Group Situation based Cooperation model [5] assume that there are some global coordinator entities which can act as control entities for other agents but it tends to make the model prone to errors and create bottlenecks in a large-scale environment. These issues show a necessity for researching new means of building distributed agent cooperation environment.

M. Klusch, M. Thimm, and M. Paprzycki (Eds.): MATES 2013, LNAI 8076, pp. 152–165, 2013.

In the current paper we propose an approach which is based on agent capabilities, that is we associate each agent with set of services it provides, so that multiagent system can be seen as distributed network of service providers. Dating back to the days when service-oriented computing was only gaining its importance, it was stated that every system which contains interacting services must ensure that the following fundamental processes are implemented in a correct and efficient way. First, a service must be described usually using structures like keywords or property-value pairs. Second, a service must declare its bindings and interfaces, as well as other information that allows clients to invoke it. Third, there must be a mechanism which allows for publishing all that information making a service discoverable. And lastly, service discovering facilities must exist, namely, query construction, routing and execution, as well as results propagation back to the requesting party. Again, since multiagent systems have inherently distributed nature of, it is natural that managing the services which are provided by each agent should be done in a decentralized way. Peer-to-peer (P2P) based approach has already established a good reputation for storing and managing content in a decentralized way, so it seems appropriate to use it for the distributed service discovery. That is, nodes in the P2P overlay network, where each node stores a description of some service in the network, act not only as routing providers and data location services, but also as servers providing service access.

Despite the fact that P2P approach is successfully used for content storage and management for about 15 years, service management systems based on it are still quite sparse and, as a general rule, they either remain within academia or even don't evolve further than proof-of-concept stage. The earliest successful P2P-based service discovery networks include Hypercube[6] and Speed-R. Also, many of proposed solutions are based on Semantic Web approach for describing services. For instance, in the approach described in [7] services are described using DAML-S while service publishing and discovery mechanism based on JXTA technology. Similar approach is used in [8], but Gnutella P2P overlay is used instead of JXTA. In [9] authors propose to perform service search according to business aspect of services and not based on service descriptions itself. This approach aims to facilitate better service composition during service execution phase later. On the other hand, [10] presents a DHT-based P2P overlay service network with partial keywords and wildcards matching support. In this network authors use a special kind of DHT key mapping - locality preserving mapping based on Space Filling Curves.

In this paper we propose a Web Service discovery architecture based on structured P2P overlay network (namely, Chord [11]), which utilizes platform-independent attribute-value based method for describing high-level properties of web services and elaborate algorithm for service discovery that allows variety of queries including range and fuzzy ones. This system is loosely based on the PWSD (Peer-to-Peer based Web service discovery) architecture presented in [12] and can be seen as an extension and enhancement of this approach. PWSD system concerns web service discovery in a P2P environment with the implementation all of four fundamental processes mentioned above. It is a lightweight approach that uses attribute-value based service description without resorting to complex data description frameworks (i.e. Semantic Web). It is based on Chord DHT and by extending it only a little contributes greatly to

simplicity and modularity of the approach. The similarity of the goals and common points in the ways to reach them are the main reasons why exactly PWSD was chosen as the starting point of our solution.

Main differences between the approach taken in PWSD and the system proposed in this article can be summarized as follows. First, while NVTree structure introduced in PWSD and (a,v)-graph structure used in this paper bear a lot of similarities, our approach do not use any explicit serialization format for the service descriptions (like XML-based WSDL format used in PWSD) and the algorithms for building abstract service description itself differ considerably. More detailed comparison between NVTree and (a,v)-graph is provided later in this article. Second, the approach to service descriptions storing taken in this paper naturally allows range and fuzzy queries which is not possible in PWSD system. And finally, while PWSD architecture includes an extension of Chord overlay network (named XChord), we show that actually there is no need for such extension to be based on some particular overlay network and different structured P2P overlays could be used together or interchangeably.

While being inherently a framework that allows people collaborate on providing, discovering and using services, the approach presented in this paper is designed in a way that makes it possible to employ intelligent agents at some (or all) nodes. That is, according to one of the definitions, which states that multiagent system is a "system in which several interacting, intelligent agents pursue some set of goals or perform some set of tasks" [13], the network for service sharing and discovery can delegate some of its activities to an agent which resides in overlay node and is capable of performing such tasks as negotiation and distributed scheduling minimizing human user participation in them. In fact, we regard this research direction as the most promising way to expand our system to real applications.

The rest of the article is organized as follows. Section 2 presents the overview of PWSD network. Section 3 present the architecture of proposed service sharing and discovery network including service description, service storing and service discovery mechanisms. Section 4 presents the experimental results. Conclusions are given in the section 5.

2 Web Service Discovery in PWSD

Given that we have a valid service description, we obtain a set of keys from it and pass them to the hash function such as MD5 to generate a set of Hash IDs (HID) which are used to locate appropriate peers by means of Peer-to-Peer routing algorithms. Subsequently, HIDs are published to the target peers, which cache the description in router repositories thus completing the publishing process. The process of service locating is roughly the same. Thus, the key step in service publishing and locating process is looking up a peer node according to HID, which was made possible by extending routing algorithm of Chord DHT.

Each peer in PWSD acts as a service peer (SP), which not only provides Web service access, but also acts as a peer in the Peer-to-Peer overlay network. Several logical machines can share one piece of hardware as well as it's possible for a single logical machine to consist of several physical machines. Each logical machine consists of three active components (Web Service Discovery Interface component, core

component, router) and one passive component called local repository. The roles and structure of these components are described in detail in [12].

Service locating algorithm specifies how to route the requests to the service peers who satisfy service request conditions. In PWSD, the service request is expressed in XML, since it's consistent with XML-based service descriptions stored in the destination service peers. However, the routing algorithm, Chord, in underlying Peer-to-Peer overlay network only supports exact match queries within each service description keyword. That's why authors presented an extension to Chord algorithm called XChord which supports XML based conditional match. In PWSD, WSDL is used to describe the Web service interface, and the service description is generated based on the content of WSDL document and the description that user inputs before publishing.

Service description generation in PWSD is based on the structure called NVTree (node-value tree). More precisely, an XML based tree node extraction approach is used for generating service descriptions. Example service description and its corresponding node-value tree can be found in the original paper [12]. It is worth noticing that only significant elements in service description will be extracted and inserted into the NVTree, and only the meaningful nodes in the NVTree will be used to generate a hash value, which in turn will be used as a hash key for service description and will be inserted into P2P overlay network. In PWSD, simple node-splitting method is used to extract each node-value pair to form NVTree and independently map them onto a key. However, only the leaf nodes in NVTree have a pair of node and value. Furthermore, in order to preserve the hierarchical relationship, the parent node of the leaf node is also extracted. Those nodes whose values consist of several words are further divided into single word value based nodes. After splitting the NVTree into separated simple description nodes, they are concatenated as strings, which are, in turn, passed to the hash function to produce hash IDs. These hash IDs are used as keys to insert into the underlying P2P overlay using XChord algorithm.

To search for a Web service, the client specifies query conditions by composing an XML document. Also, conditions can be combined using logical operators (or, and) to form composite queries. The query is processed in a way similar to the service description processing. That is, the XML document is transformed to the NVTree and then to simple description nodes. These nodes are concatenated as strings and hash ID is obtained using the same hash function from the underlying P2P overlay network. Since those hash IDs are, in fact, the keys in the underlying P2P overlay, usual DHT-based search is performed to answer the query. Finally, search results are further combined according to the logical operators used (set union for logical and, set intersection for logical or).

3 Service Sharing and Discovery Network

3.1 Overview

The idea of building service sharing and discovery network based on P2P overlay itself is not innovative, since it allows for avoiding many problems that arise in centralized scenarios like single point of failure, poor scalability or lack of robustness.

Nevertheless, even nowadays most of P2P-based systems deal with simple content sharing, which is fundamentally different from functionality of sharing services. Still, there is a range of problems in common that are present in both cases, most crucial of them being appropriate descriptions of items shared (content or services) as well as creating flexible and efficient search functionality that provide results relevant to the users criteria as much as possible. PWSD system described in the previous part of this article was designed to provide a solution to both these problems and it does so to some extent. From the other hand, authors of that approach made several architectural decisions which actually make the system less flexible in terms of basic problems mentioned above, namely service descriptions and service discovery. In this article we try to eliminate the shortcomings of the PWSD system by introducing more elaborate and platform-independent approach for describing services and formulating search queries.

The pivotal point of the system proposed is original platform-independent format of service descriptions. It is similar to the separate node descriptions obtained from NVTree in PWSD, but more exactly, it is based on Intentional Name System naming approach described in detail in [14]. Unlike PWSD, where service descriptions are initially given in XML format, the approach in our system is based on abstract graph-based attribute-value description of services which are called (a,v)-graph. The underlying serialization format of the graph is actually not important since it is not the part of the framework itself. We also propose several ways for building description graph, including utilization of existing descriptions, automatic description building and manual description input.

Despite the fact that NVTree and (a,v)-graph have much in common, it is the differences that make (a,v)-graph structure more flexible. In order to get a better view on advantages of the approach proposed in this paper we need to outline those differences more clearly. As can be seen from the way of building the NVTree, it is basically a tree representation of a XML-based (actually, WSDL-based) service description which is further split into atomic attribute-value pairs, where value nodes contain typed data (in case type information is not available, the value is assumed to be of string type). Besides, the information present in NVTree comprises all data found in the original service description, including the routing, binding and service owner information about the service itself. On the other hand, while (a,v)-graph uses the same attribute-value structure, it serves as basic format for the service description itself, which, among other ways, could be obtained by transforming corresponding service descriptions, including XML-based ones. Besides, (a,v)-graph structure contains only information that is meaningful for the service discovery, and also can be easily extended to contain the value type specification, so search queries could be executed in a type-aware way. Also it is important to note that service owner and binding information are included in the (a,v)-graph as a special node which is essential to the further execution of discovered services.

Another difference concerns the point of how those attribute-value pairs are used in the hashing process. The approach in PWSD is as follows - attribute and value string

are concatenated and passed as an argument to the hash function which determines the place where this piece of information is stored in the overlay network. The approach in the (a,v)-graph is similar but instead of hashing attribute and value altogether, the hash is computed only for attribute node and corresponding subgraph of the (a,v)-graph is copied to the responsible node according to obtained hash value. This way the structure stored at the responsible is still an (a,v)-graph, which allows for the queries to be much more flexible, which includes fuzzy and range queries. Flexibility of the queries is also stipulated by the fact that actual mechanism of (a,v)-graph storage is not defined, so different implementations can choose the best one for the specific needs.

The next step for services discoverability after descriptions is the way how they are stored in a distributed manner in P2P overlay network. Similarly to PWSD we chose Chord overlay for this purpose, but while the authors of PWSD decided to extend Chord DHT algorithm, scheme for storing and discovering service descriptions proposed in this article is overlay-independent. That is, the algorithms of storing, removing, updating and locating the services are defined in the layer completely disconnected from the DHT layer and deal with only with abstract notions like "responsible node". This way it is possible to have multiple layers of overlays for service storage (for instance, to increase reliability or distribute the load), and those overlays, in fact, do not have to be the same. This approach will be described in more detail in the next sections. Finally, we present abstract format of querying the distributed database of service descriptions which makes possible wide range of conditions including fuzzy conditions, range queries and complex values lookup.

Table 1. Key differences between NVTree and (a,v)-graph

NVTree	(a,v)-graph
• is a result of WSDL document transformation • there is no distinction between binding properties and properties of a service itself • hash value is based on concatenated attribute-value pair • responsible node is storing only a hash value like a piece of content • supports only exact match queries within given keyword	• is basic abstract format for storing service properties • contains binding information as a special node • hash value is based solely on the attribute • a structure that responsible node is storing is also an (a,v)-graph • supports fuzzy and range queries

3.2 Service Description

Each service in the network is described using attribute-value format, forming so called (a,v)-pairs, where attribute stands for the arbitrary property of a service. While attributes can obviously be only of string type, values are not required to be of string type only, so they can be of any type which is, in principle, queryable, serializable and can be efficiently stored by the underlying DHT storage mechanism. So, for example, as it is shown on the Fig. 1, value can be an array (like 'location' attribute, which contains latitude and longitude values) or even an arbitrary object (like 'availability' attribute). One more significant difference from the approach taken in PWSD is that all (a,v)-pairs form a connected graph, called (a,v)-graph, which always includes one extra node, that represents routing and binding information for the service itself. In real life situations it's not rare when one node provides access to several services, so each service is described using (a,v)-graph.

One of the most challenging issues for any new approach or framework is to provide compatibility with existing technologies, which are already widely used. It is important to note again that (a,v)-graph approach is just an abstract model of service description, therefore it should be seen as an output of some model transformation function. In our system the possible ways of obtaining the resulting (a,v)-graph model are as follows:

Manual Input. The simplest case where user enters all service description information manually, usually with some kind of GUI, but batch information input, for instance, by uploading the file with multiple attribute-value pairs is also possible. Regardless of how the data is put into the system, it always can be processed and transformed to the underlying (a,v)-graph serialization format

Transformation of Existing Service Description. To address the issue of compatibility the system must have a way to obtain (a,v)-based descriptions from existing ones. In our case this is achieved by applying necessary mode transformations, exact nature of which depends on the representation of existing models. But since in most cases existing services are described using XML-based industry standards like WSDL or OWL-S, Extensible Stylesheet Language Transformations (XSLT) language seems to be the most appropriate choice for model transformation in this case, due to its high expressive power and ability to output virtually any kind of data format using XML data as input. It is important to note that in most cases existing service description need to be transformed to (a,v)-based one only partly, since low-level details of a service (like protocol name, input parameters enumeration, IP address etc) are generally not searched upon. However, those low-level details can still be present in (a,v)-graph in the service description node to facilitate the process of binding and routing when the service is actually used.

Automatic Augmentation. Among service description properties there are often ones that are highly important as a search criteria but normally are neither supposed to be

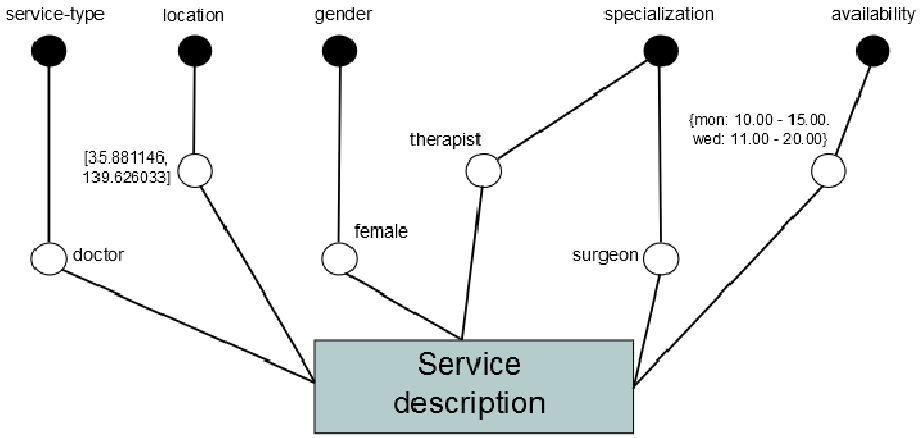

Fig. 1. (a,v)-graph for service description

input by humans nor present in traditional static service descriptions. Examples of such properties are current geographical location of the service, status of the job queue, various QoS data (like performance or latency) and sharing network data (like current node reputation). All those data can actually be obtained automatically using various means, including automatic location detection, internal monitoring functionality or network monitoring and QoS protocols.

3.3 Service Description Storing

In the approach, proposed in this article, Chord DHT peer-to-peer overlay is used to store service descriptions in a distributed manner. As usual, in order to store content in DHT we need to define what will act as a key and what exactly will be stored at nodes in the overlay. In our system, we apply hash function to each attribute name and decide the node in the overlay which is responsible for this attribute - in case of Chord DHT it is successor of a key. In terms of our approach, this node is called

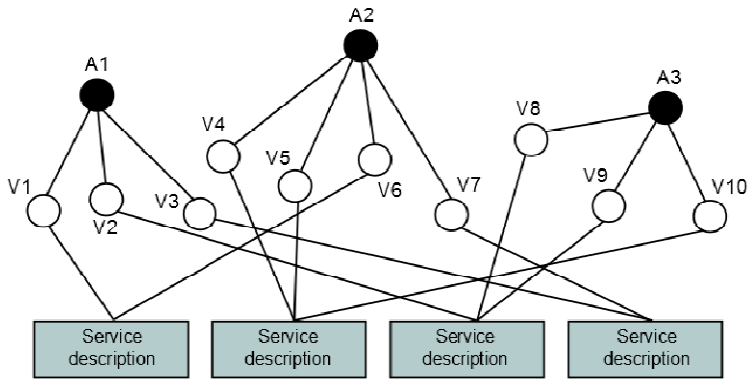

Fig. 2. Merged (a,v)-graph example

responsible for the attribute and therefore will store subgraphs of a form *[attribute, value, service description]*, that is, subgraphs based on given attribute, of all (a,v)-graphs in the network. In the result, we obtain a structure called merged (a,v)-graph in each node that is responsible at least for one attribute. Example of merged (a, v)-graph is shown in Fig. 2.

Next we present formalized version of service description storing algorithm. Each peer P in the overlay owns two graphs, namely, $P.OS$ – graph for the services it owns, and $P.SS$ – graph for the services from other peers it stores. The formal definition of both graphs is as follows: $P.OS = \left(V_V \cup V_A \cup \{sd\}, E_{A,V} \cup E_{V,SD}\right)$ and $P.SS = \left(V_V \cup V_A \cup SD, E_{A,V} \cup E_{V,SD}\right)$, where $\boldsymbol{V_V}$ – set of nodes that correspond to the attributes of the service, $\boldsymbol{V_A}$ – set of nodes that correspond to the values of the attributes, $\boldsymbol{sd}$ - node which contains the low-level service description (including an information for the owner node), $\boldsymbol{SD}$ – set of $\boldsymbol{sd}$ nodes, $\boldsymbol{E_{A,V}}$ – set of edges $(\boldsymbol{v_A}, \boldsymbol{v_V}) | \boldsymbol{v_A} \in \boldsymbol{V_A}, \boldsymbol{v_V} \in \boldsymbol{V_v}$ and $\boldsymbol{E_{V,SD}}$ – set of edges $(\boldsymbol{v_V}, \boldsymbol{sd}) | \boldsymbol{v_V} \in \boldsymbol{V_v}, \boldsymbol{sd} \in \boldsymbol{SD} \wedge \boldsymbol{sd} = \boldsymbol{sd}$. Then, we assume that for each peer in the overlay the following two functions are defined: $\boldsymbol{h}$ - hash function used to build an overlay, and $\boldsymbol{find}$ - function that returns the node from the overlay by hash value. The pseudocode for service description storing algorithm is shown in Fig. 3.

It's not essential to the architecture of the system how exactly merged (a,v)-graphs are stored at the node, but storage mechanism should be chosen in a way which makes possible answering complex queries like comparison, substring checks, set operations and so on. Therefore, the most appropriate choices for storage mechanism are relational databases or document-oriented databases, both of which have their own advantages and disadvantages.

```
foreach service s from P.OS
 foreach v_A ∈ s.V_A
   H := h(v_A);
   P := find(H);
   if not exists v_A* ∈ P.SS.V_A where v_A* = v_A
     P.SS.V_A := P.SS.V_A ∪ {v_A};
   end if
   sd* := get sd* from P.SS.SD where sd* = s.sd;
   if sd* is NULL
     P.SS.SD := P.SS.SD ∪ {s.sd};
     sd* := s.sd;
   end if
   V := {v_V ∈ s.V_V | (v_A, v_V) ∈ s.E};
   foreach v ∈ V
     P.SS.V_V := P.SS.V_V ∪ {v};
     P.SS.E := P.SS.E ∪ {(v_A, v)};
     P.SS.E := P.SS.E ∪ {(v, sd*)};
   end foreach
 end foreach
end foreach
```

Fig. 3. Service description storing algorithm

3.4 Service Discovery

Among one of the most significant drawbacks of DHT-based P2P overlays is that the principles of content storage and its association with a key usually allows only for exact query conditions when searching. Alternative, there are some elaborate approaches addressing this issue which, but again they usually offer only wildcard matching. The approach we propose in this article is based on the way service descriptions are stored in the overlay network, that is, using merged (a,v)-graph.

Firstly, we assume that each search query in the system is submitted in the form, shown in Fig. 4 or can be represented as such. Here we let $op_i(A^i)$ be some operators (=, *!=, contains,* >, < etc) defined on the sequence of attributes A_i as parameters, A_i - sequence of attributes $[a_1,a_2,...,a_n]$ (where n is arity of the operator op_i.) and lop_i denote logic operators *OR* or *AND*. This format itself is very generic, so we think that it represents most of meaningful search queries submitted in P2P networks. You can see a concrete example of the query in Fig. 5.

$$op_1(A_1) \quad lop_1 \quad op_2(A_2) \quad lop_2 \quad \ldots \quad lop_{n-1} \quad op_n(A_n)$$

Fig. 4. Generic search query format

```
service-type = 'doctor'
AND
(location = 'Tokyo' OR location = 'Yokohama')
AND
availability > 'June 15th, 2012, 2:00PM'
AND
availability <= 'June 17th, 2012, 9:30AM'
```

Fig. 5. Example of search query

Algorithm of search query routing and execution is formalized below. Note that the algorithm actually doesn't require any extensions for the underlying DHT algorithm. The query issued by the peer in the form shown in Fig. 4 can be represented as a graph $\boldsymbol{QU} = (\boldsymbol{Q} \cup \boldsymbol{LOP}, \boldsymbol{E})$, where $\boldsymbol{Q} = \{\boldsymbol{q_i} = (\boldsymbol{a_i}, \boldsymbol{op_i}) | \boldsymbol{i} = \boldsymbol{1} \ldots \boldsymbol{n}\}$ - the set of query terms, $\boldsymbol{LOP} = \{\boldsymbol{lop_i} | \boldsymbol{i} = \boldsymbol{1} \ldots \boldsymbol{n}\}$ – the set of logical operators and $\boldsymbol{E} = \cup_{\boldsymbol{i=1}}^{\boldsymbol{n}}\{\{(\boldsymbol{q_i}, \boldsymbol{lop_i})\} \cup \{(\boldsymbol{lop_i}, \boldsymbol{q_{i+1}})\}\}$. In addition to peer functions $\boldsymbol{h}$ and $\boldsymbol{find}$ introduced in the section 3.3, we assume that each peer have function $\boldsymbol{evaluate}$ which returns the result of evaluating a query term again internal database of the peer. Similarly, we define graph $\boldsymbol{RES} = (\boldsymbol{R} \cup \boldsymbol{LOP}, \boldsymbol{E})$, where $\boldsymbol{R} = \{\boldsymbol{r_i} | \boldsymbol{i} = \boldsymbol{1} \ldots \boldsymbol{m}\}$ - initially empty set of results obtained from peers after evaluating a query term, $\boldsymbol{LOP}$ is the same as $\boldsymbol{QU}.\boldsymbol{LOP}$ and $\boldsymbol{E} = \cup_{\boldsymbol{i=1}}^{\boldsymbol{n}}\{\{(\boldsymbol{r_i}, \boldsymbol{lop_i})\} \cup \{(\boldsymbol{lop_i}, \boldsymbol{r_{i+1}})\}\}$. Also we define the set of pairs $\boldsymbol{C} = \{(\boldsymbol{OR}, \cap), (\boldsymbol{AND}, \cup)\}$ which shows the one-to-one correspondence between logical operators and set-theoretical operations. Pseudocode for the algorithm is shown in Fig. 6.

```
foreach q ∈ QU.Q
  H := h(q.a);
  P := find(H);
  R := P.evaluate(q);
  RES.R :=RES.R ∪ {R} for corresponding lop ∈ LOP;
end foreach
return r_1 C[lop_1] r_2 C[lop_2] ... C[lop_{n-1}] r_n
          where r_i ∈ RES.R, lop_i ∈ RES.LOP;
```

Fig. 6. Search query routing and execution algorithm

4 Results

The implementation of the framework described above has been done using Chord overlay and consists of the following main modules: Chord overlay, the service storage, statistics and operational information collection module and the visualizer. To demonstrate the system operation we chose a domain which consists of three types of entities, namely hotels, museums and baseball matches, which act like services to be stored and queried in the P2P-based services network. The attributes (and data types of respective values) used to describe each type of service are shown in the table 2.

Given services are then put into the service storage network according to the procedure described in the section 3.3, that is, a hashing function defined in the underlying Chord overlay is applied to each attribute and according to its value the attribute

Table 2. Service types and attributes for the service P2P network example

Hotel	Museum	Baseball match
• name • suite type (single or double) • price for the one night stay • meals (yes/no field) • number of available rooms • location	• name • type of exhibition (painting, sculpture, photograph) • entrance fee • start date of exhibition • end date of exhibition • time the exhibition opens (daily) • time the exhibition closes (daily) • location	• league • (Central, Pacific) • competing teams (pair) • venue name • price • date of the match • time the match starts • time the match ends • venue location

for the given service is stored at the according node. The storage layer itself is implemented using MongoDB, document-oriented "no-SQL" database engine, which suits best for the storage of attribute-value data due to its flexibility and relative simplicity. In the course of the implementation of the storage layer it became clear that at least three database tables per node are needed for the full coverage of basic data manipulation functions, that is (1) putting service to the storage, (2) updating service stored at the overlay, (3) removing service from the storage and (4) getting service from the storage based on the query. The tables and their structure are shown in the table 3.

Table 3. Database tables used for storing services at storage node and their structure

Local service data	Remote service data	Service attributes data
comprehensive data about given node services locally	attribute data for services in the overlay network; all attributes the node is responsible of are stored	information about responsible nodes for attributes of given service; service, responsible for storing this information, is determined by hashing service name itself;

After the data is placed in the overlay we can perform services search using the query format described earlier in this article. The sample query and result returned for it are shown in Fig. 7.

```
Waiting until the overlay stabilizes...
Populating the data storage...
Ready
get location like "Ginza" and price < "10000":int32
**************************************************
RESULTS:

============================================================
Service name: Hotel_430907
name = Hotel Gracery Ginza (Responsible node: 111 = 133.38.238.161:16912, 1)
suite-type = single (Responsible node: 40 = 133.38.238.161:19926, 1)
price = 5000 (Responsible node: 40 = 133.38.238.161:19926, 1)
meals = False (Responsible node: 105 = 133.38.238.161:19051, 1)
location = Ginza 7-10-11, Chuo-ku, Tokyo Prefecture (Responsible node: 105 = 133
.38.238.161:19051, 1)
rooms-available = 0 (Responsible node: 166 = 133.38.238.161:17157, 1)
Service owner: 111 = 133.38.238.161:16912, 1
============================================================



============================================================
Service name: Hotel_655822
name = Hotel Gracery Ginza (Responsible node: 111 = 133.38.238.161:16912, 1)
suite-type = single (Responsible node: 40 = 133.38.238.161:19926, 1)
price = 6000 (Responsible node: 40 = 133.38.238.161:19926, 1)
meals = True (Responsible node: 105 = 133.38.238.161:19051, 1)
location = Ginza 7-10-11, Chuo-ku, Tokyo Prefecture (Responsible node: 105 = 133
.38.238.161:19051, 1)
rooms-available = 6 (Responsible node: 166 = 133.38.238.161:17157, 1)
Service owner: 111 = 133.38.238.161:16912, 1
============================================================
```

Fig. 7. Sample query and result

As to the evaluation of the approach proposed in this paper, we compare it to PWSD approach in terms of query flexibility. As was already stated earlier, in current approach, the way descriptions are stored in the overlay and the way queries are handled allow for virtually every query underlying database can handle. Given the example query shown in Fig. 5, the way of its handling would be the following: we split the query according to the attributes used in it (in this case it would be 3 groups for attributes **service-type**, **location** and **availability** respectively), find the nodes responsible for each attribute and let them execute their part of the query against their own internal database, which is assumed to be able to handle range queries as well (that is, using operators >, < etc.). On the other hand, mechanism proposed in PWSD cannot handle the range part of this query, since, as was described in section 2, each peer stores hash value of the (attribute, value) pair, naturally allowing only exact match queries against data in the original service description.

5 Conclusions

This paper presents the application of P2P technology to solve the problem of efficient services sharing and discovering in a decentralized way. The main advantages of proposed approach are the usage of well-known Chord overlay, which guarantees the correctness and soundness of underlying P2P overlay, and the approach which allows complex, fuzzy and range queries nevertheless keeping the structured overlay approach, while modular framework architecture allows using virtually any P2P overlay and storage mechanism. From the other hand, many of open issues remain, such as introducing distributed transactions for overlay services manipulations, nodes identities that are not tied to the node address (mainly for ad hoc networks) and correct management of services for departed and newly joined nodes.

References

1. Foundation for Intelligent Physical Agents. FIPA Contract Net Interaction Protocol Specification, version H, `http://www.fipa.org/specs/fipa00029/index.html`
2. Foundation for Intelligent Physical Agents. FIPA Recruiting Interaction Protocol Specification, version H, `http://www.fipa.org/specs/fipa00034/index.html`
3. Jennings, N.R.: Controlling cooperative problem solving in industrial multi-agent systems using joint intentions. Artificial Intelligence 75(2), 195–240 (1995)
4. Pynadath, D.V., Tambe, M., Chauvat, N., Cavedon, L.: Toward Team-Oriented Programming. In: Jennings, N.R. (ed.) ATAL 1999. LNCS, vol. 1757, pp. 233–247. Springer, Heidelberg (2000)
5. Kim, M.S., Kim, M.K., Lee, J.T.: Group Situation based Cooperation Model. In: Proceedings of the 2007 International Conference on Convergence Information Technology (ICCIT 2007), pp. 1372–1377 (2007)
6. Schlosser, M.T., Sintek, M., Decker, S., Nejdl, W.: HyperCuP – Hypercubes, Ontologies and Efficient Search on P2P Networks. In: Moro, G., Koubarakis, M. (eds.) AP2PC 2002. LNCS (LNAI), vol. 2530, pp. 112–124. Springer, Heidelberg (2003)

7. Ramljak, D., Matijašević, M.: SWSD: A P2P-Based System for Service Discovery from a Mobile Terminal. In: Khosla, R., Howlett, R.J., Jain, L.C. (eds.) KES 2005. LNCS (LNAI), vol. 3683, pp. 655–661. Springer, Heidelberg (2005)
8. Paolucci, M., Sycara, K., Nishimura, T., Srinivasan, N.: Using DAML-S for P2P Discovery. In: International Conference on Web Services, ICWS 2003 (2003)
9. Hu, J., Guo, C., Wang, H., Zou, P.: Web Services Peer-to-Peer Discovery Service for Automated Web Service Composition. In: Lu, X., Zhao, W. (eds.) ICCNMC 2005. LNCS, vol. 3619, pp. 509–518. Springer, Heidelberg (2005)
10. Schmidt, C., Parashar, M.: A Peer-to-Peer Approach to Web Service Discovery. J. World Wide Web 7, 211–229 (2003)
11. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: a scalable peer-to-peer lookup service for Internet applications. In: Proceedings of ACM SIGCOMM 2001, pp. 149–160. ACM (2001)
12. Li, Y., Zou, F., Wu, Z.-D., Ma, F.-Y.: PWSD: A Scalable Web Service Discovery Architecture Based on Peer-to-Peer Overlay Network. In: Yu, J.X., Lin, X., Lu, H., Zhang, Y. (eds.) APWeb 2004. LNCS, vol. 3007, pp. 291–300. Springer, Heidelberg (2004)
13. Weiss, G. (ed.): Muliagent Systems. The MIT Press, Cambridge (1999)
14. Winoto, W.A., Schwartz, E., Balakrishnan, H., Lilley, J.: The design and implementation of an intentional naming system. In: Proceedings of the Seventeenth ACM Symposium on Operating Systems Principles, vol. 33(5), pp. 186–201. ACM (1999)