A Semantic Service Discovery Network for Large-Scale Ubiquitous Computing Environments

Saehoon Kang, Daewoong Kim, Younghee Lee, Soon J. Hyun, Dongman Lee, and Ben Lee

This paper presents an efficient semantic service discovery scheme called UbiSearch for a large-scale ubiquitous computing environment. A semantic service discovery network in the semantic vector space is proposed where services that are semantically close to each other are mapped to nearby positions so that the similar services are registered in a cluster of resolvers. Using this mapping technique, the search space for a query is efficiently confined within a minimized cluster region while maintaining high accuracy in comparison to the centralized scheme. The proposed semantic service discovery network provides a number of novel features to evenly distribute service indexes to the resolvers and reduce the number of resolvers to visit. Our simulation study shows that UbiSearch provides good semantic searchability as compared to the centralized indexing system. At the same time, it supports scalable semantic queries with low communication overhead, balanced load distribution among resolvers for service registration and query processing, and personalized semantic matching.

Keywords: Scalable semantic service discovery, ubiquitous computing, semantic search.

Manuscript received Nov. 03, 2006; revised Aug. 20, 2007.

Saehoon Kang (phone: + 82 42 866 6251, email: kang@icu.ac.kr), Daewoong Kim (email: wooni@icu.ac.kr), Younghee Lee (email: yhlee@icu.ac.kr), Soon J. Hyun (email: shyun@icu.ac.kr), and Dongman Lee (email: dlee@icu.ac.kr) are with the School of Engineering, Information and Communications University, Daejeon, Rep. of Korea.

Ben Lee (email: benl@eecs.oregonstate.edu) is with the School of Electrical Engineering and Computer Science, Oregon State University, Corvallis, Oregon, USA.

I. Introduction

In ubiquitous computing, services are dynamically searched and automatically integrated with an application according to the user's context, such as location, activity, preference, and so on [1]. With the popularization of network-enabled mobile devices and network infrastructures, and the emergence of new applications, such as m-commerce or telematics, the scale of ubiquitous computing environments has expanded from small spaces, such as u-homes or u-offices, to very large spaces, such as u-cities or u-metropolises. In such a large-scale ubiquitous computing environment, service discovery is the key factor in providing appropriate services among all the services that are spread over a wide-area network [2], [3].

Developing a service discovery system for such a large-scale ubiquitous computing environment poses several technical challenges. First, the system must be scaled to the number of available service discovery resolvers. Usually, services are registered with more than one service discovery resolver. Since a centralized index of services does not exist, these resolvers conceptually form a service discovery overlay network which allows resolvers to collaborate with each other to efficiently discover services. Therefore, as the number of distributed service discovery resolvers increases, the communication overhead among them needs to be carefully managed.

The system also needs to update the states of registered services. For example, when a traffic monitoring camera service detects changes in road traffic conditions, it should immediately update its attribute values and report the changes to the service discovery system. Load balancing is another important issue with the distributed service resolvers. The overall performance of the system will significantly degrade when frequent service (re)registrations are concentrated on

particular resolvers. Therefore, the workload for service registration must be distributed and balanced among all the resolvers as evenly as possible.

Also, the system needs to provide semantic queries in addition to the keyword-based queries to better support user-centric ubiquitous computing. Semantic search capability is important in terms of providing users with alternative services from the currently available services when exactly matching services are not available.

Finally, the system needs to reflect user preferences in ranking the candidate services based on a similarity function to achieve personalized service discovery. For example, suppose a user wants a color printing service in the vicinity, and there are two candidate printer services in the network. One is located nearby but does not support color printing capability, while the other provides color printing but is located farther from the user. The resulting service depends on the user preference as to which printer service is more suitable.

Although a number of distributed service discovery systems have been proposed in the literature, most of them are designed for small-scale domains, such as home and office [4]-[7]. Recently, there have been some efforts to address service discovery for a large-scale ubiquitous computing environment [8]-[10]. These methods have some of the previously mentioned features but not all of them. For example, INS/Twine system provides good scalability but only supports keyword-based matching [8]. Other methods [9], [10] support semantic queries, but they do not take into account frequent updates of service and registration and query load balancing.

Therefore, this paper presents an efficient service discovery scheme for a large-scale ubiquitous computing environment, called *UbiSearch*, which supports scalable semantic queries with low communication overhead and balanced load distribution for service registration and query processing. Services in the proposed scheme are defined by attribute-value pairs (AV-pairs) and the semantic of each attribute is described using ontology [11], [12]. The unique feature of UbiSearch is the use of a *semantic service discovery network* in a semantic vector space (SVS), where the services are mapped using the semantic coordinate vector assigned by their ontology-based AV-pairs. In this way, similar services are clustered in the semantic service discovery overlay network. Similarly, queries for a desired service are also mapped in the same semantic space. The cost of identifying the best matching service is minimized by searching only the cluster region around the location of the query. The size of a region to be searched is determined by the semantic distance a user can accept. By mapping the services in a semantic space, the search space for a query is efficiently confined within a minimized cluster region while keeping a high accuracy comparable to the centralized scheme. Therefore, the proposed scheme provides both scalability and semantic searchability.

The rest of this paper is organized as follows. Section II discusses related works. Section III presents the proposed UbiSearch system and its major components. In particular, the section discusses a semantic coordinate assignment scheme to map services into a SVS, a load balancing technique to evenly distribute service indexes across service discovery resolvers, and an efficient search scheme to minimize the number of resolvers to visit. The simulation study is discussed in section IV. Finally, conclusion and future work are presented in section V.

II. Related Work

Service discovery is a key function to implement context-aware applications for mobile users in ubiquitous computing environments [13]. Many service discovery systems have been proposed using a variety of architectures ranging from centralized systems to fully distributed overlay systems [4]-[10], [14]-[23].

Systems with centralized service lookup are mainly useful for small-scale ubiquitous environments, such as ubiquitous-homes or ubiquitous-offices [4]-[7], [15]. However, these systems are not appropriate for large-scale environments because they suffer from single point of failure, the server becomes a bottleneck as the registration and query rates increase, and they may become potential attack point, such as denial-of-service. On the other hand, since all the indexes are managed by a single server, it is easy to find the most relevant services using a semantic matching scheme with minimal communication overhead.

To avoid the problems of centralized approaches, a number of distributed systems have been proposed, where service indexes are distributed to multiple resolver nodes and queries are routed to the appropriate resolvers. Many of them take advantage of techniques such as a distributed hash table (DHT) and Bloom Filter developed for P2P information sharing systems [24]-[29]. They are classified as either structured or unstructured approaches based on how the resolvers are organized.

The unstructured resolver networks do not scale well with the number of services and queries as they suffer from index and query flooding [17], [18]. To reduce the network bandwidth consumption caused by flooding, some techniques, such as the use of Bloom Filter gossip information are employed [17], [18]. However, they are not sufficient to resolve scalability problems and do not guarantee searchability. (Searchability means that the relevant services can be retrieved if they are available in the network.)

In hierarchical systems, resolvers are organized in a tree structure, where parent resolvers are responsible for all the

domains managed by their child resolvers [9], [10], [19]. The domains are created based on particular attributes in the service description, such as geographical location and service type. These approaches are very useful for cases in which services are described by hierarchical names, such as DNS. However, as (state-updating) registration and query rates increase, the higher level resolvers, especially the root resolver, may become bottlenecked. If queries involve multiple orthogonal attributes, which are not considered when defining domains for each resolver, all the responsibilities are imposed on the root resolver. In the traffic monitoring service example discussed in section I, domains are created based on the geographical locations, and resolvers are organized hierarchically by streets, cities, and metropolitan levels. Therefore, queries such as "Find all the cameras filming the congested cross ways" cannot be handled in a scalable manner.

Hash-based systems use a DHT to form a self-organizing structure of resolvers [3], [14], [19]. By hashing a term, such as service name or ID, each service is associated with a specific resolver. A query message is also routed to the associated resolver using the hashed value. Due to the randomness of hash functions, DHT-based approaches have good load-balancing properties. However, the randomness of hash functions is also the main obstacle to the use of DHT-based schemes for semantic service discovery because semantic matching is not supported.

To add semantic searchability while taking the advantages of DHT-based schemes, some systems use different mapping functions between services and resolvers instead of hash functions [21], [22]. For example, the Mercury system supports range queries by partitioning the value space into buckets and mapping values and ranges to one or more buckets [21]. However, it does not consider all the attributes when ranking services based on their semantic similarity to the given query. The work closest to ours is the pSearch system [22], which distributes the indexes of documents to the content-addressable network (CAN) nodes using classical information retrieval algorithms, such as the vector space model (VSM) and latent semantic indexing (LSI). However, this technique is difficult to employ in a large-scale ubiquitous computing environment due to its inherent problems, such as processing cost, central LSI computation, and so on. The proposed UbiSearch system is different from [22] in that indexes are distributed to the semantic space in a distributed fashion using landmark-based service coordination and user's preferences are dynamically considered in choosing relevant services.

III. UbiSearch

To provide scalable semantic service discovery in a large scale ubiquitous computing environment, the service discovery system has to be able to retrieve the most semantically relevant service in the network without searching the entire space. The proposed UbiSearch system starts from a very simple idea in which semantically similar services are clustered around nearby resolvers in a service discovery overlay network so that the search space for a given query is efficiently limited.

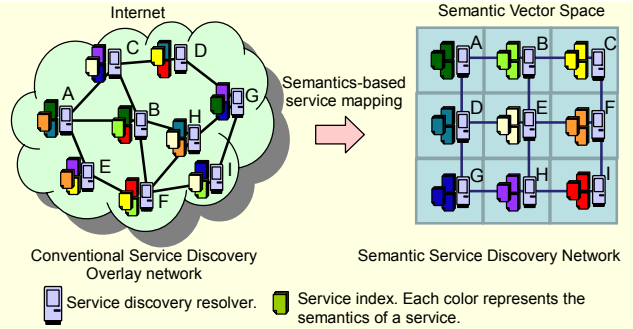


Fig. 1. Concept of UbiSearch.

Figure 1 depicts the concept of UbiSearch. The semantics of a service is represented by the color of a service index, and the semantic similarity of two services is measured by the similarity of the color of corresponding indexes. Therefore, services are clustered based on the colors of their indexes. The services with four primary colors such as green, yellow, red and blue are registered with the resolvers A, C, I, and G, respectively, which are furthest from each other because their semantics are the most different from each other. Meanwhile, service indexes with compound colors are clustered somewhere between the resolvers of primary colors. For example, the indexes with orange color are clustered between the resolvers C and I with which yellow and red indexes are registered because orange color is more similar to yellow and red than to blue and green.

To realize the proposed idea, the UbiSearch system contains the following four components: a *service description* to describe service semantics, a *semantic distance function* to measure similarity between services, a *semantic vector space* to semantically associate services and resolvers, and a *semantic service discovery network* to efficiently route query and registration messages to the most appropriate service resolver, which is responsible for semantically finding the best match for the desired service.

The following subsections describe each component of UbiSearch.

1. Service and Query Description

Services and queries can be described by various properties, such as type, attributes, functions, and so on. In a small-scale ubiquitous computing environment, only type or interface name has been used to describe a desired service, as seen in

UPnP [4], Jini [5], and Bonjour [30] because few services are of the same type in such an environment. However, as the scale of the ubiquitous computing environments becomes larger, the number of services with the same type in a domain increases. In this situation, users want to select the most appropriate service among the available services, for example, the closest printer among all the printers in a building, or the highest-definition color printer on the current floor. Therefore, a service discovery system needs to provide finer-grained discovery results, based on service type or interface name, as well as various service attributes. In a ubiquitous computing environment, however, there is no guarantee that an exactly matching service will be available. When there is no exact mach, the service discovery system needs to return a service that is most similar to the desired service through semantic matching to allow applications to run without disruptions.

The descriptions of a service and a query in UbiSearch are represented as $D = (a_1v_1, a_2v_2, \cdots, a_nv_n)$, where a_i and v_i $(1 \le i \le n)$ represent an attribute and its value, respectively [8], [14]. For example, attributes of a multimedia streaming service may include title, picture size, required bandwidth, and so on.

Figure 2 shows an example description for a traffic monitoring camera service, which contains the type information and three AV-pairs, including *location = Broadway*, *traffic state = busy*, and *road condition = dry*. Note that only the boldfaced descriptions are used for service discovery and *type* is a special attribute.

The attributes are assumed to be orthogonal because if two attributes are dependent on each other, that relation must be reflected in the ontology hierarchy. For example, if there were two attributes *city* and *street* instead of a single attribute *location*, these two attributes are dependent on each other; that is, Broadway is included in New York City. However, this relation is already expressed in the location ontology hierarchy. Therefore, the two attributes, city and street, can be merged into a single attribute, location.

```
<service>
    <id> TC1004</id>
    <type>
        <name> Traffic Monitoring Camera </name>
    </type>
    <attributes>
        <attribute>
            <name> location </name>
            <value> Broadway </value>
        </attribute>
        <attribute>
            <name> traffic state </name>
            <value> busy </value>
        </attribute>
        <attribute>
            <name> road condition </name>
            <value> dry </value>
        </attribute>
    </attributes>
    <interfaces>
        …
    </interfaces>
    …
</service>
```

Fig. 2. Service description for traffic monitoring camera.

In order to support semantic queries, each value of the AV-pairs is described in a well-defined ontology, which provides information about the semantic relations among the concepts. There has been a great deal of ontology research reported in an effort to support semantic service discovery [3], [9], [10], [15], [31], [32]. Out of various semantic relations in ontology, the two most essential semantic relations, *'is-a'* and *'part-of'* relations, are used. Thus, each value is assumed to have either of these two relations. Moreover, the ontology trees are assumed to be shared by all the nodes including service resolvers, service providers, and clients. Thus, nodes are aware of which type of ontology should be used to describe the services to be registered or queried.

Note that it is also possible to use other properties, such as provider information, access protocols, and so on, which can be described in a service description file. The service semantics is described as attributes by service developers, and access protocols can also be described as one of the attributes provided the service developer which is deemed necessary to describe the service semantics. If two services are of the same type, they have the same attribute sets.

2. Semantic Distance Function

In order to determine the semantic distance between two services, a similarity function first needs to be defined between two concepts, c_1 and c_2, which represent attributes in an ontology tree. In a prior work, several similarity measuring strategies were compared based on a hierarchically structured semantic network similar to the ones shown in Fig. 3 [33]. They show that the following similarity measure, $\mathrm{Sim}(c_1, c_2)$, gives the best results:

$$\mathrm{Sim}(c_1, c_2) = \begin{cases} e^{-al} \cdot \dfrac{e^{\beta h} - e^{-\beta h}}{e^{\beta h} + e^{-\beta h}} & \text{if } c_1 \neq c_2, \\ 1 & \text{otherwise,} \end{cases} \quad (1)$$

where l is the length of the shortest path between c_1 and c_2 in the semantic tree, and h is the level of the tree of the direct common subsumer from c_1 and c_2, and $\alpha \ge 0$ and $\beta \ge 0$ are parameters scaling the contribution of the shortest path length and depth, respectively. According to their benchmark tests, the optimal parameters are $\alpha = 0.2$ and $\beta = 0.6$. In this paper, we do not focus on how to design a similarity function. The optimal similarity function can be different depending on application domains. The purpose of UbiSearch is to retrieve the relevant services to the given query when an appropriate similarity function is given.

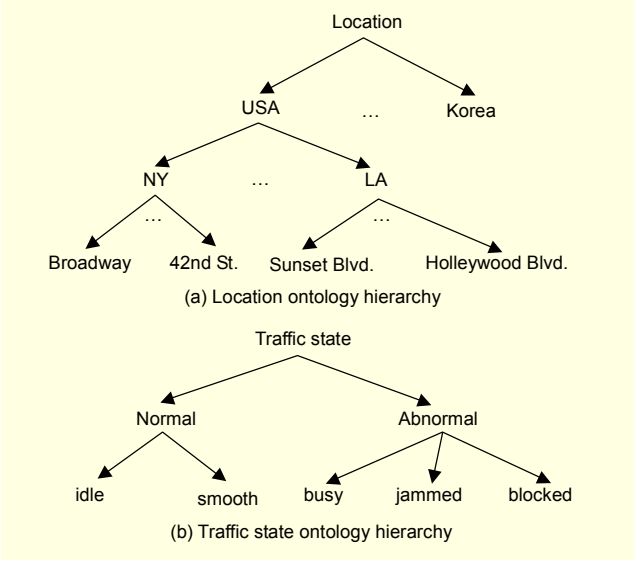


Fig. 3. Examples of ontology hierarchy for locations and traffic states. The arrows represent subsumption relationships.

The intuition behind (1) is that the semantic similarity is governed by the depth of the subsumer as well as the shortest path between two concepts. This is because the concepts at upper levels in a hierarchically structured semantic network aremore general and semantically less similar than concepts at lower levels. Therefore, the semantic distance function between two services S_1 and S_2 is defined as

$$D(S_1, S_2) = \frac{1}{n}\sum_{i=1}^{n} \omega_i (1 - \mathrm{Sim}(v_{i1}, v_{i2}))^2, \qquad (2)$$

where n is the number of attributes, v_{i1} and v_{i2} are the values of attribute a_i of service S_1 and S_2, respectively, and ω_i is the weighting factor for attribute a_i given by a user during query request where $\sum \omega_i$ is n.

In practice, the similarity between two services is determined based on user preferences. For instance, when measuring the similarity between two printer services, some users may consider the locational proximity of printers to be the most important, while other users may consider printing speed to be the most important. By defining a weighting factor ω_i for each attribute of a desired service, users can apply their preferences to measure similarity. However, when locating a service in a vector space, the semantic distance function $D(S_1,S_2)$ with all $\omega_i = 1$ should be used because every user should share the common vector space.

3. Semantic Vector Space

In order for semantically similar services to be registered with nearby resolvers, a semantic association method is needed to link a service and its corresponding resolver according to its semantics. As a semantic association tool between them, we introduce an n-dimensional vector space called SVS in which the similarity between two services can be approximated by measuring the distance between the two points, that is, the semantic distance of two services is proportional to the distance.

Services and queries are assigned to the points in SVS by the semantic mapping function. The semantic mapping function should semantically map similar services onto nearby positions in SVS; that is, the more similar two service's semantics, the closer the corresponding positions. This will be described in more detail in section II.5.

4. Semantic Service Discovery Network

UbiSearch uses a semantic service discovery network (SSDN) to efficiently route query and registration messages. Given a semantic query, an SSDN has to be able to retrieve the most semantically relevant service in the network without searching the entire space. To achieve this goal, SSDN provides several novel mechanisms for registering services with nearby resolvers in the network and efficiently routing the registration and query messages with high scalability.

The CAN, which is a P2P network with a DHT [24], is used to organize the resolvers and route messages. A DHT-based network has many advantages, such as high scalability, load balancing, logarithmic hop routing, fault tolerance, and self-organizing capability [24]-[28]. A DHT provides the same functionality as a traditional hash table by storing the mapping between a key and a value. Using hash-table interfaces *put* (*key, value*) and *get* (*key*), values are stored at the live overlay node(s) and retrieved from the overlay node(s) where values can be objects of any type. We chose CAN because it supports multi-dimensional space, which is the underlying space for the proposed method. However, UbiSearch is different from CAN in that it organizes the services in the vector space based on their semantic distance instead of randomly organizing them. Moreover, resolver nodes are bootstrapped using the coordinates of a local service as a bootstrapping key rather than a random key. Note that UbiSearch system only uses CAN as a message routing tool.

The resolvers are located in an n-dimensional virtual vector space with each having its own distinct zone in the space. The zones split or merge whenever resolvers join or leave. A service is assigned coordinates in the vector space based on its semantics. Semantically similar services are mapped onto nearby positions; thus, the more similar two services' semantics are, the closer the corresponding positions in SVS. A service is registered with a resolver if the coordinate of the service falls into the resolver's zone.

Services and queries are routed to a resolver according to their semantic coordinates. Figure 4 shows an example of service registration and search in SSDN. When a client

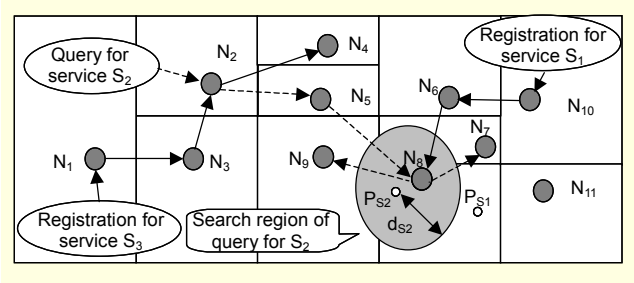


Fig. 4. Service registration and search in a two-dimensional semantic vector space.

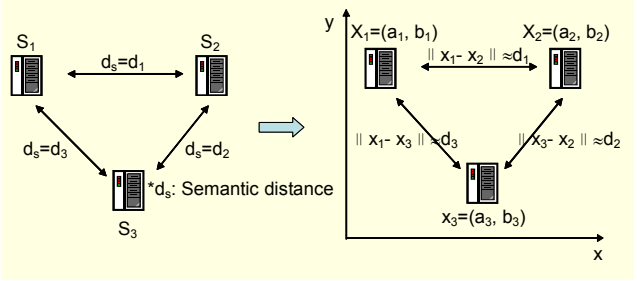


Fig. 5. Semantic coordination.

requests registration for service S_1 with the resolver N_{10}, N_{10} generates the semantic coordinate of S_1 and its corresponding point, P_{S1}, and routes the registration message to the corresponding position. The registration message for service S_1 is routed to the resolver N_8, which manages the zone that includes P_{S1}. Then, the resolver N_8 stores the index of service S_1 and is responsible for providing information about it. Similarly, the query message for service S_2 with semantic distance d_{S2} is routed to node N_8, which is also responsible for the zone containing point P_{S2} of S_2 as depicted the shaded region in Fig. 4. When the resolver N_8 fails to find the appropriate services, the query message is routed to an adjacent resolver, namely, N_7 or N_9 in this example, which shares the same region with radius d_{S2} from P_{S2}.

The value of d_{S2} is determined by the maximum dissimilarity a user can accept. When a user requests service discovery, a value d_{S2} ($0 \le d_{S2} \le |SVS|$) which specifies the service dissimilarity to the desired service S_2 is defined. Services with values greater than d_{S2} are implied to be useless and, thus, neglected. The distance between two points in SVS is counter-proportional to the similarity between the two services by the nature of SVS. Therefore, a service can be identified without searching the entire space.

The following subsections provide detailed descriptions of the generation of semantic coordinates for services, the distribution of service registration and the query processing load, and an efficient search scheme to minimize the number of service resolvers that need to be visited.

5. Semantic Coordinates

The basic idea behind semantic coordination is to assign each service a coordinate value, such that the Euclidean distance between two coordinates approximates the semantic distance between the two services. Using this scheme, services that are semantically close to each other are mapped to nearby positions in SVS.

Figure 5 shows the measured semantic distances $d_1 = S_1 - S_2$, $d_2 = S_2 - S_3$, and $d_3 = S_1 - S_3$. Based on these distances, a two-dimensional vector set is assigned to each service as a set of coordinates. The semantic distance d_S is approximated by the Euclidean distance: $d_s(S_1, S_2) \approx \| x_1 - x_2 \|$.

The problem of calculating a coordinate based on the inter-object distances (in our case, the semantic distances among services) is called the *embedding* problem [34], which is defined as follows.

- A metric space is defined by (X, d), where X is a set, and d is a metric, that is, a distance function which satisfies symmetry, positive definiteness, and triangle inequality.
- A Euclidean space R^n is a metric space (Y, δ), where Y is a vector set and δ is the Euclidean norm, that is, $d(x_1, x_2) = \| x_1 - x_2 \|$.
- An embedding is a mapping $\varphi: X \rightarrow R^n$.

Given some X, d, and n, we seek an accurate embedding, that is, a φ with $\delta(\varphi(x_1), \varphi(x_2)) \approx d(x_1, x_2)$ for all x_1 and x_2 in X. The challenge is to minimize errors between the measured distance and estimated distance.

Although various embedding schemes have been proposed [35]-[39], we apply Lipschitz embedding for semantic service coordination due to its simplicity and speed [34]. It is based on the idea that two close objects in a metric space are typically located at similar distances from other objects so that two nearby points in the original metric space have very similar coordinate vectors.

In order to utilize Lipschitz embedding for a semantic service coordination, a set $L \subset X$, where X is a service set and $L = \{l_1, l_2, \cdots, l_n\}$, is first defined. Services in the set L are called *landmark services* and every service type has its own landmark service set defined by a landmark generation function whose inputs are ontology trees for each attribute of a service type. For a given distance function d, the Lipschitz embedding with respect to L is the mapping, such that $\varphi(x) = [d(x, l_1), d(x, l_2), \cdots, d(x, l_n)]$, where $|L| = n$, and the mapping is onto the vector space R^n, where n is dimensionality of a coordinate vector. In other words, to obtain the coordinate vector $x_i \in R^n$ for service S_i, the j-th component of x_i is set to the measured distance between service S_i and landmark service j, for $j=1, \cdots, n$. In general, the accuracy of the coordination improves as the number of landmarks increases.

Figure 6 shows an example of Lipschitz embedding for

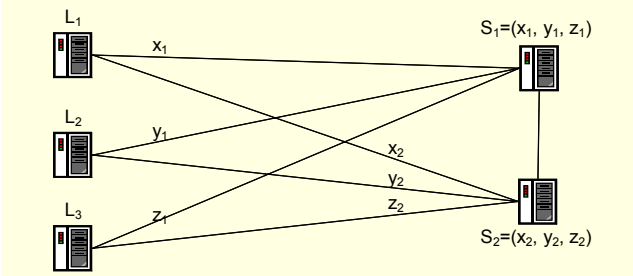


Fig. 6. Lipschitz embedding for service coordinate.

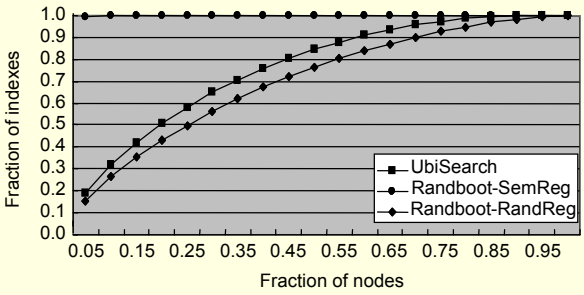


Fig. 7. Index distribution of UbiSearch.

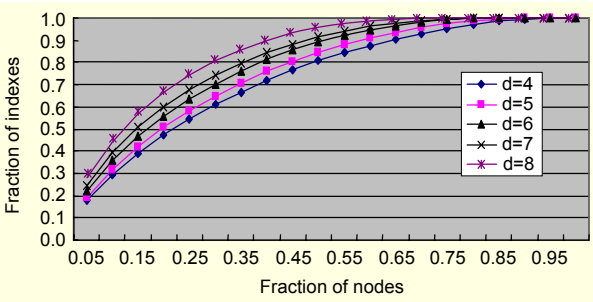


Fig. 8. Effect of index distribution as varying dimensionality of the semantic space.

service coordination when $n = 3$, where L_1, L_2, and L_3 are a set of landmark services for a given service type of either S_1 or S_2, and S_1 has distances x_1, y_1, and z_1 to landmark services L_1, L_2 and L_3, respectively. Similarly, S_2 has x_2, y_2, and z_2. The distance is measured using a semantic distance function D defined by (2). As previously mentioned, the semantic distance function $D(S_1,S_2)$ with all $\omega_i = 1$ is used because every user should share the common vector space.

The distance sets (x_1, y_1, z_1) and (x_2, y_2, z_2) are assigned to service S_1 and S_2 as coordinates. If S_1 and S_2 are very similar, they will have similar distance sets from the services L_1, L_2 and L_3; in other words, the distance sets (x_1, y_1, z_1) and (x_2, y_2, z_2) are very similar.

6. Balanced Index Distribution

In order for resolvers in SSDN to evenly share the service registration and query processing load, service indexes need to be evenly distributed among the resolvers. In CAN, the nodes are randomly bootstrapped and the indexes are registered with a node using keys generated by a hash function, which evenly disperses service indexes among all the resolvers in the network.

However, service registration using semantic coordinates may lead to unbalanced index distribution. The coordinates of a service tend to be mapped towards the middle of the entire vector space because they are generated based on the distances to a set of landmarks as shown in (2).

To resolve unbalanced index distribution, the local service's coordinates are used as a bootstrapping key instead of the random bootstrapping key used in CAN. Thus, a resolver node that joins the UbiSearch network randomly selects one of the locally registered services and calculates its coordinates. The resolver node then sends a join-request message containing the coordinates. A join-request message is routed according to the coordinates, and finally, it arrives at a UbiSearch node in a zone defined by the coordinates. After receiving the join-request message, the resolver node divides its zone into two smaller zones and assigns one of the zones to the join-request sender.

As this process is repeated, the size of zones is adjusted to the density of indexes, and the denser the index population of a region, the smaller zones into which the region is split.

Figure 7 shows the index distribution of UbiSearch for a network of 5 dimensions and 5,000 nodes. The number of service indexes is 50,000. Nodes are arranged in decreasing order according to the number of indexes they have.

The Randboot-RandReg curve shows the index distribution when random node bootstrapping and random registration are used and implies the ideal index distribution case. Due to the randomness, the curve tends to be round-shaped. However, as the number of indexes increases, the shape of the curve becomes straighter. For the Randboot-SemReg line, nodes are randomly bootstrapped and a service index is registered according to its semantic coordinates. In this case, 5% of the nodes contain 98% of the indexes. For UbiSearch, nodes are bootstrapped according to the coordinates of a randomly selected local service, and this results in a similar shaped curve as that in Randboot-RandReg.

We observe that our local service based on node bootstrapping provides similar load balancing improvement compared to the original CAN. Figure 8 shows the effect of index distribution as the dimensions of the semantic space vary with 5,000 nodes. The index distribution becomes unbalanced because the size of each zone becomes larger as the dimensions of the semantic space increase, so the populated zone cannot be subdivided into smaller zones. In general, the high

dimensionality improves the discovery accuracy, and at the same time, it may lead to unbalanced index distribution.

7. Minimization of Search Space

In this section, we describe the mismatch problem between semantic distance in SVS and semantic similarity on the user side. This causes the system to visit more resolvers to retrieve semantically matching services. We also describe a query propagation technique to reduce the number of resolvers to visit.

A. Mismatch between Semantic Distance and Semantic Similarity

UbiSearch uses semantic distance functions with different weights to build a semantic vector space and measure the similarity between a user's query and registered services. A SVS is built based on the distances to a set of landmarks according to (2), where the user preference for each attribute remains the same, that is, all $\omega_i = 1$. However, since users have their own preferences regarding each service attribute, the semantic distance function D for measuring service similarity will have different weighting factors ω_i for different users. During the query request, a user assigns a weighting factor ω_i for each attribute ranging from 1 to 10 according to his/her preferences in measuring semantic similarity. Then, the weighting factors are normalized so that $\sum \omega_i$ is n.

Although there is a strong relationship between the similarities of two services and the distance in SVS, there is a mismatch because different distance functions are used to measure similarity on the user side. This means that even if two services are located at the same distance from the querying resolver node in SVS, the measured similarities on the user side could be very different. This requires the search range to be expanded, increasing the number of resolvers that need to be visited. The following subsection describes how to reduce the number of resolvers to visit to find the services relevant to the given query.

B. Reducing the Number of Resolvers to Visit

There is a trade-off between the number of nodes to visit and the number of nodes with which services are actually stored. When services in a network frequently join and leave, requiring the states to be frequently updated, multiple service registrations cause a large overhead in terms of network bandwidth and (re)registration message processing at each resolver node. In order to reduce the number of resolvers to visit, the following two techniques are adopted to discover semantically matching services: one-hop replication [23] and intelligent query propagation.

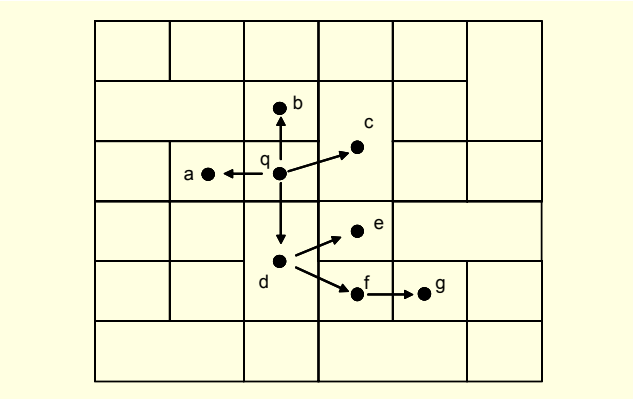


Fig. 9. Example of query spreading.

The one-hop replication scheme allows a service index to be registered with multiple resolver nodes. This is done by first registering an index with a revolver and then forwarding the registration to its one-hop neighbor resolvers. This way the indexes are shared among one-hop neighbors, and the number of resolvers to visit can be reduced because the neighbor resolver nodes that have better indexes can be identified without visiting them.

The intelligent query propagation technique addresses the problem of efficiently routing query messages to find the services in the network. When a query message arrives at a resolver node, the search process starts by routing the query message to its neighbor nodes until the service is identified. If the search range is expanded in a ring fashion, the number of nodes visited increases exponentially as the dimensions of the search space increases. To resolve this problem, the relevance of services in the given node is used to determine how a query message should be forwarded.

Figure 9 shows an example of query propagation, where the rectangles represent the zones in an SVS with each zone containing a resolver node. Suppose that a user wants to find the top-k services that are most similar to the query description. A set of weighting factors ω_i for the semantic distance function is attached to the query message to define similarity based on user preference.

The search starts at node q whose zone is defined by the coordinates of the query. The query message is forwarded to its one-hop neighbor nodes a, b, c, and d. Each of these nodes ranks the stored indexes according to the semantic distance function based on the user defined weighting factors ω_j. The neighbor nodes a to d send top-k indexes to node q with their neighbor list. Note that nodes a to d have their own indexes, their neighbor's indexes, and indexes that have been shared among one-hop neighbors. Then, node q aggregates the returned indexes with its own and selects new top-k indexes.

If the returned indexes from the neighbor nodes overlap with the new top-k generated by node q, the query message is first forwarded to the neighbor's neighbors who hold the overlapping indexes. For example, since node d has overlapping indexes, which are registered with nodes e and f, the query message is forwarded to nodes e and f. When top-k indexes from nodes e and f are returned to node q, a new set of top-k indexes is generated again and the process repeats. The message is not sent to nodes that have already received the same message. The search process terminates when the new set of top-k indexes is better than the previous top-k, and the final top-k results are returned to the user.

IV. Simulation

In order to study the performance of UbiSearch, a simulator was developed using PlanetSim, an object-oriented simulation framework for overlay networks and services [40]. PlanetSim was developed in Java language and it clearly distinguishes between overlay algorithms, such as key-based routing and the applications built on top of them.

For the simulation study, the following three metrics are observed: the accuracy of search results compared with the centralized indexing scheme, the number of visited resolvers as a function of dimensions of the semantic space, and the efficiency of service ranking reflecting user preference.

1. Simulation Environment

The simulation study assumes that 2,000 resolvers form an SSDN. Two sets of indexes of sizes 20,000 and 100 are used to specify the number resources and queries, respectively. The number of AV-pairs for a resource description varies from 4 to 7. Each attribute can take 2^{l+1}-2 values from the corresponding ontology tree whose concepts are organized as a binary tree with depth l = 4, 5, 6. The number of landmarks used for dimensionality reduction varies between 3 and 10. The descriptions of landmark services are randomly generated, except for the value taken from the leaf node of the ontology hierarchy tree, in order to distribute the landmark coordinates as evenly as possible. The top 20 services from the centralized indexing scheme are assumed to be relevant. The returned number of services varies from 10 to 40. The weighting factor ω_i for each attribute in semantic distance function is randomly generated from a range of 1 to 10.

Table 1 shows the parameters used in the simulation. The attribute values for queries are assumed to be evenly distributed to each concept level in the ontology hierarchy because the users may use unspecific concepts for a query during a semantic search.

Table 1. Simulation parameters.

	Description	Value
N	Number of resolvers	2,000
s	Number of services	20,000
a	Number of AV-pairs for a service description	4 to 7
l	Depth of ontology tree	4 to 6
v	Number of values of each attribute	2^{l+1}–2
d	Number of dimensions	3 to 10
q	Number of queries	100
k	Number of relevant services	20
r	Number of returned services	5 to 30
ω_i	Weighting factors for each attribute	1 to 10

2. Evaluation for Accuracy

The accuracy of search results is defined by two factors, precision and recall. Precision is defined as the ratio of the number of relevant indexes retrieved to the total number of irrelevant and relevant indexes retrieved. On the other hand, Recall is defined as the ratio of the number of relevant indexes retrieved to the total number of relevant indexes in the system.

If all the service indexes are stored in a single resolver in a centralized manner, they are sorted in increasing order according to the given semantic distance function and query. The top-k indexes are regarded as the indexes for the relevant services, that is, semantically matched services. These indexes are referred to as set A. Then, j indexes are retrieved for the same semantic distance function and query using UbiSearh. These returned indexes are referred to as set B. The recall (R) and precision (P) are then defined as

$$R = \frac{|A \cap B|}{|A|}, \qquad P = \frac{|A \cap B|}{|B|}.$$

A. Varying the Number of Returned Services

We first examine how the recall and precision change as a function of the number of returned services. In the simulation, services are described with four attributes and the depth of ontology trees for each attribute is five. We use the number of returned services $r = \{5, 10, 15, 20, 25, 30\}$ and $k = 20$. The top 20 services are assumed to be returned from the centralized resolver and are regarded as a relevant service set.

Figure 10 shows the percentage of recall as a function of the number of returned services r and the network dimension d. As

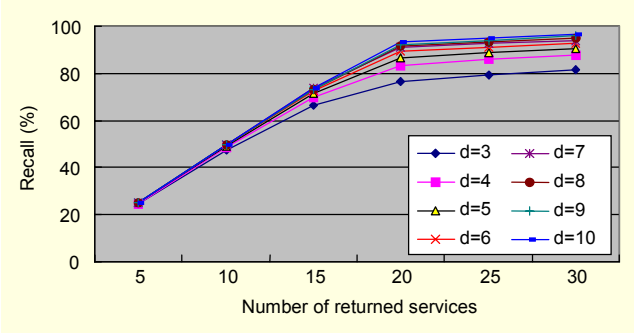


Fig. 10. Comparison of recalls as function of the number of returned services with $a = 4$ and $l = 5$.

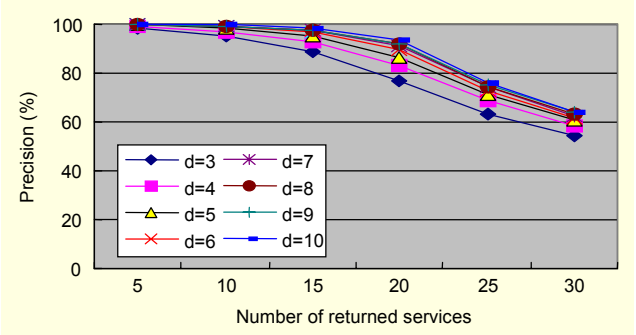


Fig. 11. Comparison of precisions as function of the number of returned services with $a = 4$ and $l = 5$.

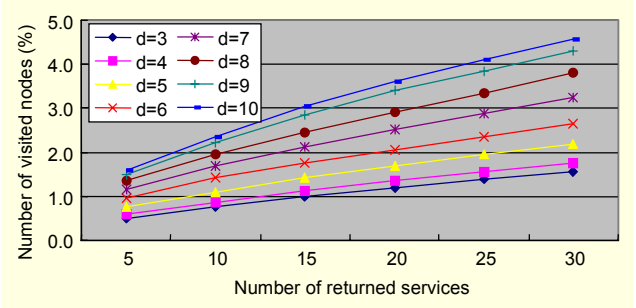


Fig. 12. Comparison of the numbers of visited nodes as function of the number of returned services with $a = 4$ and $l = 5$.

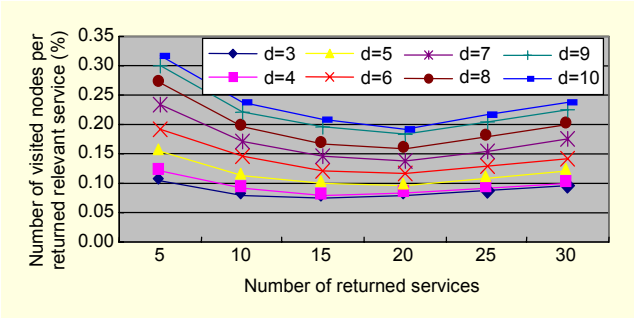


Fig. 13. Comparison of the numbers of visited nodes to return one relevant service as function of the number of returned services with $a = 4$ and $l = 5$.

can be seen, the percentage of recall increases sharply when r is small and starts to level off after r reaches k. Figure 11 shows precision as a function of varying r and d. Contrary to recall, precision decreases very slowly as r increases until r reaches k and then decrease sharply beyond k. These results show that when $r < k$, most of the returned services are relevant services. This means r does not need to exceed k unless a user wants to find a large number of relevant services. When $r = 5$, the precision for all dimensions is over 99%. When $r = 20$ and $d = 7$, both precision and recall are 91.1%. As a reference, recall and precision are usually inversely related. If $k = r$, the values of precision and recall are the same by definition. Therefore, the term accuracy is used here to mean both precision and recall when $k = r$.

There is a strong relationship between the number of visited nodes and the accuracy of the discovery results. Figure 12 shows the effect of returned services on the number of visited nodes. As r increases, the number of visited nodes also increases linearly. This is because as r becomes larger, the possibility for the neighbor nodes to have top-r services becomes higher and the query is forwarded to them. As the number of dimensions increases, the number of visited nodes increases because the number of neighbor nodes also increases.

Meanwhile, the number of visited nodes required to return one relevant service decreases as r increases. This is illustrated in Fig. 13. When $d = 7$, the average number of visited resolver nodes to return 5 services is only 1.17% of the total resolver nodes. On average, 0.23% of the total resolver nodes need to be visited to return one relevant service. Almost all of the returned services are relevant services. When r increases to 20, 2.52% of the total nodes are visited and 0.14% of the total nodes are needed to return one relevant service. However, only 91.1% of the returned services are relevant. If the number of returned services exceeds the number of relevant services, the average number of visited nodes to return one relevant service increases again.

B. Varying the Number of Attributes and Ontology Depth

We next evaluate the accuracy as varying the number of service attributes a and ontology depth l. Figure 14 shows the effect on the accuracy when varying a. As a increases, the accuracy decreases. Figure 15 shows that as the depth of the ontology tree increases, the accuracy also decreases. As a and l become larger, the more sharply the accuracy decreases. This is because when d is constant, the quality of semantic coordination is dependent on the number of possible descriptions of a service. Therefore, the accuracy decreases sharply as a and l increase. Note that the number of possible service descriptions increases exponentially when a or l

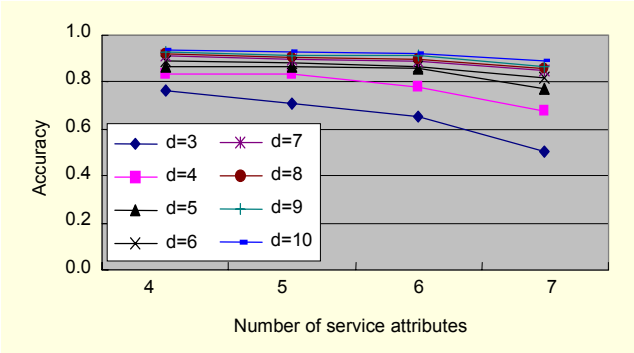


Fig. 14. Accuracy comparison as function of the number of service attributes with $l = 5$.

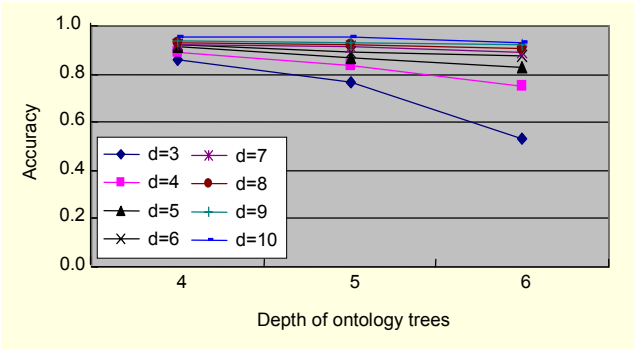


Fig. 15. Accuracy comparison as function of the depth of ontology trees with $a = 4$.

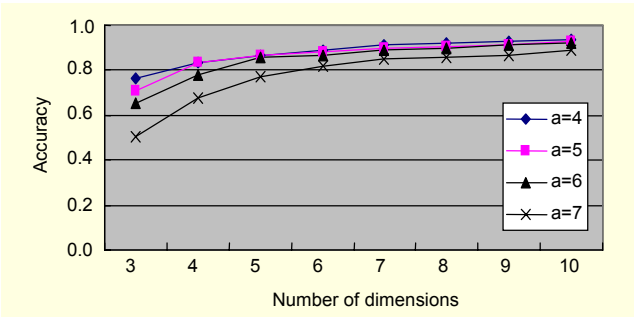


Fig. 16. Accuracy comparison as function of the number of dimensions with $l = 5$.

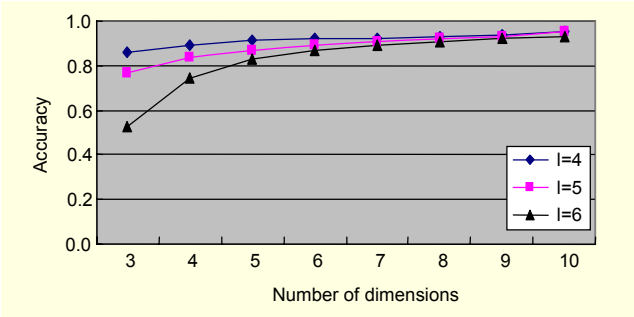


Fig. 17. Accuracy comparison as function of the number of dimensions with $a = 4$.

increases by 1. As d increases, the degree of accuracy decrease is reduced. For instance, the accuracy decreases from 95.5% to 93.1% when $d = 10$ and $a = 4$, and l changes from 4 to 6, but it decreases from 86.0% to 52.9% when $d = 3$ and $a = 4$. Similarly, when $d = 10$ and $l = 5$ and a changes from 4 to 7, the accuracy decreases from 93.5% to 89%. The accuracy decreases from 76.4% to 50.2% when $d = 3$.

C. Varying the Number of Dimensions

Figures 16 and 17 compare the accuracy when varying the number of dimensions with $l = 5$ and $a = 4$. Actually, they are made with the same data as in Figs. 14 and 15 to show how the accuracy improves as d increases; however, the accuracy increases quickly at first, and the degree of improvement gradually decreases. When d is greater than 7, there is only a very small improvement. This implies that the optimal number of dimensions is around 7. We obtained similar results from simulations with different parameters.

3. Evaluation for Query Forwarding Scheme

We evaluated the performance of our query forwarding scheme based on the service relevance for the purpose of reducing the number of nodes to be visited. To validate the efficiency of the proposed scheme, we compared it with the ring search scheme in terms of visited nodes until the same accuracies were achieved. Figure 18 shows the accuracy variations as the number of visited nodes increases using the ring search scheme. It seems that the accuracies are almost the same regardless of the number of dimensions used in the UbiSearch overlay network configuration; however, on closer examination, we can see that the accuracy improves little by little as the number of dimensions decreases. This is because as the number of dimensions increases, the more neighbor nodes are located within similar distances while a service is mapped onto SVS more accurately.

Figures 10 to 12 show that when $d = 7$ and $r = 20$, we

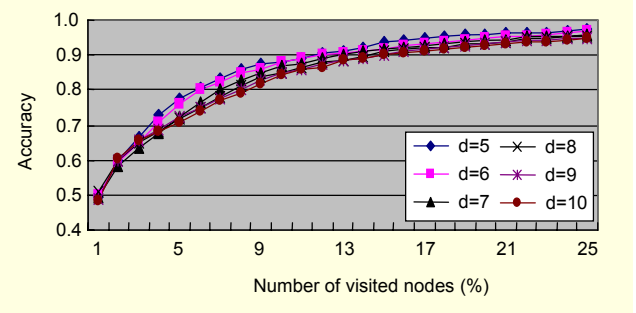


Fig. 18. Accuracy comparison as function of the number of visited nodes using the ring search scheme with $a = 4$ and $l = 5$.

achieve 91.1% accuracy by visiting only 2.52% of the total nodes using the proposed scheme. Figure 18 shows that the ring search scheme requires 14.3% of nodes to be visited to obtain the same accuracy. This means that the proposed scheme reduces the number of visited nodes by 17.6% compared to that of the ring search scheme. We also obtain similar results with different parameters.

V. Conclusion and Future Work

This paper presented an efficient semantic service discovery scheme for large-scale ubiquitous computing environments which supports scalable semantic queries with low communication overhead, balanced load distribution among resolvers for service registration and query processing, and personalized semantic matching.

Services are semantically defined using ontology-based attribute-value pairs and the semantic similarity between them is measured by the personalized semantic distance function. We modeled a semantic service discovery network on the semantic vector space where the services are mapped using a semantic coordinate vector assigned by the landmark-based semantic coordination scheme. The semantic space is divided into semantic zones which are allocated to the resolvers. Services are registered with a resolver, which contains their semantic coordinates. Similarly, queries for a desired service are also mapped in the same semantic space. Thus, services semantically close to each other are mapped to nearby positions so that the similar services are registered in a cluster of resolvers. Using this mapping technique, the search space for a query is efficiently confined within a minimized cluster region while keeping a high accuracy comparable to the centralized scheme.

We also proposed techniques to evenly distribute service indexes to the resolvers and reduce the number of resolvers to visit. Our simulation results demonstrated that UbiSearch provides good semantic searchability comparable to a centralized indexing system while maintaining scalability, low communication overhead, load balancing among resolvers, and preference-based semantic matching.

Our future work will include the use of other similarity measuring techniques and ontology sets in the proposed system. Investigation of the effect of skewed service distribution and query patterns as well as enhancement of our scheme for MANET would be also important.

References

[1] M. Weiser, "The Computer for the Twenty-First Century," *Scientific American*, Sept. 1991, pp. 94-10.

[2] S. Vinoski, "Service Discovery 101," *IEEE Internet Computing*, Jan./Feb. 2003, pp. 69-71.

[3] C. Sangpachatanaruk and T. Znati, "A P2P Overlay Architecture for Personalized Resource Discovery, Access, and Sharing over the Internet," *Consumer Communications and Networking Conference (CCNC'05)*, Jan. 2005.

[4] Universal Plug and Play Forum, Universal Plug and Play Device Architecture, v. 0.91, http://www.upnp.org, Mar. 2000.

[5] Sun. Technical White Paper: Jini Architectural Overview, http://www.sun.com/jini/, 1999.

[6] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, v. 2," http://www.efceditor.org/rfc/rfc2608.txt, 1999.

[7] Salutation Consortium, White Paper: Salutation Architecture: Overview. http://www.salutation.org/whitepaper/originalwp.pdf, 1998.

[8] M. Balazinska, H. Balakrishnan, and D. Karger, "INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery," *Proc. of International Conference on Pervasive Computing,* Zurich, Switzerland, Aug. 2002, pp. 195-210.

[9] K. Verma, K. Sivashanmugam, A. Sheth, A. Patil, S. Oundhakar, and K. Miller, "METEOR-S WSDI: A Scalable Infrastructure of Registries for Semantic Publication and Discovery of Web Services," *Information Technology and Management*, vol. 6, no. 1, 2005, pp. 17-39.

[10] T. Pilioura, G. Kapos, and A. Tsalgatidou, "PYRAMID-S: A Scalable Infrastructure for Semantic Web Service Publication and Discovery," *Proc. of 14th International Workshop on Research Issues on Data Engineering: Web Services for e-Commerce and e-Government Applications*, 2004, pp. 15-22.

[11] T.R. Gruber, "A Translation Approach to Portable Ontology Specification," *Knowledge Acquisition*, vol. 5 no. 2, 1993, pp. 199-220.

[12] N.F. Noy and D.L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology," *Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, Mar. 2001.

[13] F. Zhu, M.W. Mutka, and L.M. Ni, "Service Discovery in Pervasive Computing Environments," IEEE *Pervasive Computing*, vol. 4, no. 4, Oct.-Dec. 2005, pp. 81-90.

[14] J. Gao and P. Steenkiste, "Design and Implementation of a Distributed Scalable Content Discovery System," *IEEE Journal on Selected Areas in Communications,* vol. 22, no. 1, Jan. 2004, pp. 54-66.

[15] D. Chakraborty et al., "DReggie: Semantic Service Discovery for M-Commerce Applications," *Proc. of Workshop on Reliable and Secure Applications in Mobile Environment*, Oct. 2001.

[16] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, J. Lilley, "The Design and Implementation of an Intentional Naming System,"

Proc. 17th ACM SOSP, Kiawah Island, SC, Dec. 1999.

[17] F.M. Cuenca-Acuna, C. Peery, R.P. Martin, and T.D. Nguyen., "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities," *International Symposium on High Performance Distributed Computing*, June 2003.

[18] S.C. Rhea and J. Kubiatowicz, "Probabilistic Location and Routing," *IEEE INFOCOM*, 2002.

[19] T.D. Hodes, S.E. Czerwinski, B.Y. Zhao, A.D. Joseph, and R.H. Katz, "An Architecture for Secure Wide-Area Service Discovery," *Proc. of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, Aug. 1999, pp. 24-35.

[20] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat, "Scalable Wide-Area Resource Discovery," *UC Berkeley Technical Report UCB//CSD-04-1334*, July 2004.

[21] A. Bharambe, M. Agrawal, and S. Seshan, "Mercury: Supporting Scalable Multi-Attribute Range Queries," *SIGCOMM 2004*, Portland, Aug. 2004, pp. 353-366.

[22] C. Tang, Z. Xu, and S. Dwarkadas, "Peer-to-Peer Information Retrieval Using Self-Organizing Semantic Overlay Networks," *ACM SIGCOMM 2003*, Germany, Aug. 2003, pp. 175-186.

[23] C. Gkantsidis, M. Mihail, and A. Saberi, "Hybrid Search Schemes for Unstructured Peer-to-Peer Networks," *Proc. of INFOCOM*, 2005.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A Scalable Content-Addressable Network," *ACM SIGCOMM'01*, Aug. 2001, pp. 161-172.

[25] A. Rowstron and P. Druschel, "Pastry: Scalable, Distributed Object Location and Routing for Large-Scale Peer-to-Peer Systems," *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*, Nov. 2001, pp. 329-350.

[26] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *SIGCOMM'01 Symposium on Communications Architectures and Protocols*, Aug. 2001, pp. 149-160.

[27] B.Y. Jhao, J.D. Kubiatowicz, and A.D. Joseph, "Tapestry: An Infrastructure for Fault-Tolerant Wide-Area Location and Routing," *Technical Report UCB/CSD-01-1141*, UC Berkeley, 2001.

[28] Frank Dabek, "A Distributed Hash Table," Ph.D. thesis, Massachusetts Institute of Technology, Nov. 2005.

[29] B. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, July 1970, pp. 422-426.

[30] Apple Inc, Bonjour in Mac OS X, http://www.apple.com/macosx/features/bonjour/

[31] W3C Semantic Web, available at http://www.w3.org/2001/sw/

[32] F. Heine, M. Hovestadt, and O. Kao, "Towards Ontology-Driven P2P Grid Resource Discovery," *Proc. of 4th IEEE/ACM International Workshop on Grid Computing (GRID'04)*, 2004, pp. 76-83.

[33] Y. Li, Z.A. Bandar, and D. McLean, "An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, July/Aug. 2003, pp. 871-882.

[34] L.Y. Tang and M. Crovella, "Virtual Landmarks for the Internet," *Proc. of the ACM/SIGCOMM Internet Measurement Conference*, 2003, pp. 143-152.

[35] J. Bourgain, "On Lipschitz Embedding of Finite Metric Spaces in Hilbert Space," *Israel Journal of Mathmatics*, vol. 52 no.1-2, 1985, pp. 46-52.

[36] I. Borg and P.J.F. Groenen, *Modern Multidimensional Scaling,* 2nd ed., Springer, New York, 2005.

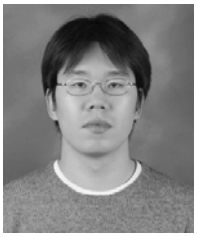
[37] T.S. Eugene Ng and H. Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," *Proc. of Infocom*, June 2002.

[38] M. Costa, M. Castro, A. Rowstron, and P. Key, "PIC: Practical Internet Coordinates for Distance Estimation," *International Conference on Distributed Computing Systems*, Tokyo, Japan, Mar. 2004.

[39] M. Pias, J. Crowcroft, S. Wilbur, T. Harris, and S. Bhatti, "Lighthouses for Scalable Distributed Location," *Proc. of International Workshop on Peer-to-Peer Systems*, Feb. 2003.

[40] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo, "PlanetSim: A New Overlay Network Simulation Framework," *Lecture Notes in Computer Science (LNCS)*, vol. 3437, Mar. 2005, pp. 123-137.

Saehoon Kang received the BS degree in computer science from Korea University, Seoul, Korea, in 1995. He received the MS and PhD degrees in computer engineering from Information and Communications University, Daejeon, Korea, in 1999 and 2007, respectively. From 1995 to 1998, he worked for LG Semicon Ltd. as a research engineer. He was a postdoctoral researcher at Carnegie Mellon University, Pittsburgh, PA, from March to August in 2007. He is currently a postdoctoral researcher at Cleveland State University, Cleveland, OH. His research interests include ubiquitous computing, resource discovery, mobile ad hoc networks, and overlay networks. Dr. Kang is a member of the IEEE and IEICE.



Daewoong Kim receives the BS degree in computer engineering in 2005 from Hankook Aviation University, Korea, and the MS degree in computer engineering in 2007 from Information and Communications University, Daejeon, Korea. He is a research engineer at Mediachorus Inc. His interests are in the area of mobile and pervasive computing, with a specific focus on service discovery.

Younghee Lee received the BS and MS degrees in electronics engineering from Seoul National University, Seoul, Korea, in 1976 and 1980, respectively. He received the PhD degree in computer science from Universite de Technologie de Compiegne, France, in 1984. From 1984 to 1997, he worked at Electronics and Telecommunications Research Institute (ETRI) as a research team leader and as a director of the Protocol Engineering Center. From 1997 to 2000, he served as a vice chairman of the ITU-T SG7, and a chairman of the WP3. He was a visiting scientist at IBM T. J. Watson Research Center from 1986 to 1987. From 1998 to 2001, he served as the Dean of School of Engineering at Information and Communications University (ICU), Korea, and he is now a professor at ICU.

Soon J. Hyun received the BS degree in electrical engineering from the Kyungpook National University, Daegu, Korea; the ME degree in electrical engineering from the Katholike Universitate Leuven, Havelee, Belgium; and the PhD degree in electrical and computer engineering from the University of Florida, in 1981, 1987, and 1995, respectively. His PhD dissertation was on parallel query processing in object-oriented temporal database systems. He is an associate professor of the School of Engineering, Information and Communications University (ICU), Korea. From 1983 to 1997, he worked with the Electronics and Telecommunications Research Institute (ETRI), Daejeon, Korea, where he was leading a digital library development project in an effort to build the Korea National Information Infrastructure. From 1984 to 1986, he was a visiting member of the research staff at Bell Telephone/ITT (presently, Bell/Alcatel), Antwerp, Belgium, where he worked on the development of telecommunications network protocol systems. His recent research interests include context management, ubiquitous computing, temporal database management, multimedia databases and knowledge-based information search over information highway, and digital library systems, and applications development.

Dongman Lee received the BS degree in computer engineering from Seoul National University, Seoul, Korea, in 1982, and the MS and PhD degrees in computer science from KAIST, Daejeon, Korea, in 1984 and 1987, respectively. From 1988 to 1997, he worked as a technical contributor at Hewlett-Packard. From 1998 to Nov 2003, he was an associate professor in School the of Engineering, Information and Communications University (ICU), Daejeon, Korea. Since December 2003, he has been a professor at ICU. He has been actively participating in Korean and international Internet number and name committees since 1998. He received a Prime Minister Award in recognition of his contribution to the advancement of the Korean Internet in 2000. He received the Internet Technical Achievement Award in KRNet07 in 2007. He has served as a member of the program committee of numerous international conferences including IEEE Multimedia, COMPSAC, PDCS, PRDC, VSMM, ICAT, and so on. He is an Editor of JCN. His research interests include distributed systems, computer networks, mobile computing, and pervasive computing. Dr. Lee is a member of KISS, IEEE, and ACM.

Ben Lee received the BE degree in electrical engineering from the State University of New York (SUNY) at Stony Brook, New York, USA, in 1984, and the PhD degree in computer engineering from the Pennsylvania State University, University Park, in 1991. He is currently an associate professor in the School of Electrical Engineering and Computer Science at Oregon State University. He has been on the program committees and served as publicity chair for several international conferences. Currently, he is chairing the 2008 International Workshop on Pervasive Wireless Networking (PWN08). His research interests include wireless networks, computer system architecture, multithreading and thread-level speculation, and parallel and distributed systems. Dr. Lee is a member of the IEEE.