Concept-Based Discovery of Mobile Services

Chara Skouteli
Department of Computer Science,
University of Cyprus
CY-1678 Nicosia, Cyprus
chara@ucy.ac.cy

George Samaras
Department of Computer Science,
University of Cyprus
CY-1678 Nicosia, Cyprus
cssamara@ucy.ac.cy

Evaggelia Pitoura
Department of Computer Science,
University of Ioannina
GR 45110, Ioannina, Greece
pitoura@cs.uoi.gr

ABSTRACT
In this paper, we consider semantic service discovery in a global computing environment. We propose creating a dynamic overlay network by grouping together semantically related services. Each such group of services is termed a community. Communities are organized in a global taxonomy whose nodes are related contextually. The taxonomy can be seen as an expandable distributed semantic index over the system services, which aims at improving service discovery. Our performance results indicate that in certain cases, our service discovery mechanism outperforms even the case in which service indexes are fully replicated at all system sites.

Keywords
Mobile computing, pervasive computing, service discovery.

1. INTRODUCTION
Nowadays, a significant amount of data is stored on a variety of small devices, such as smart phones, palmtops and personal computers. These small devices are inter-connected, thus composing a global network that is characterized by (i) device heterogeneity, (ii) large-scale data distribution, (iii) data heterogeneity, (iv) device mobility, and (v) a variety of communication protocols. Data stored on these small diverse devices creates what we call a *global* or *universal* database. The goal of our research is to provide both the theoretical foundations and the system infrastructure for effectively querying this database [7]. To overcome differences in the communication protocols used by mobile devices and data and device heterogeneity, we employ a *service oriented* approach in that data are wrapped and accessed through web services. In this paper, we focus on the fundamental issue of how to efficiently query for services in such a global database.

We propose creating a dynamic overlay network above the core system of web services to group together semantically related services, thus creating a network of *communities*. Each of these communities is a set of references to semantically related services that are distributed over the global mobile environment (for example, a community of weather services, or a community of services related to music). Communities are distributed; they are organized in a global taxonomy whose nodes are related semantically. This taxonomy can be seen as an expandable, flexible and distributed semantic index over the system services, which aims at improving the cost of service discovery. In addition, we support the notion of context. Context is used to constraint the number of the semantically related services to those that are appropriate for a given context. We model context as a set of (attribute, value) pairs.

Our performance results indicate that community-based service discovery works well under various workloads. In particular, in wireless environments and for a service request rate of 200 requests/second, it provides performance improvements even over an approach in which each site can satisfy every service request locally.

The remainder of this paper is organized as follows. Section 2 introduces concept-based discovery and communities. Section 3 describes the community overlay network, while Section 4 highlights query processing. Performance evaluation is presented in Section 5. Section 6 discusses related work and Section 7 presents conclusions and future work.

2. CONCEPT-BASED DISCOVERY
DBGlobe [7] is a middleware platform for mobile computing, that provides support for describing indexing and querying services offered by a large number of geographically dispersed small mobile devices. DBGlobe employs a *service-oriented* approach in that all the data and resources offered or requested by the small devices are accessed through services. We adopt a hybrid (partially ad-hoc) architecture where geographical 2-D space is divided into adjacent administrative areas (similar to GSM cells) each managed by a Cell Administration Server (CAS) (see Figure 1). In our current design, each cell represents the area of coverage of a network access point. Each CAS provides low-level functionality, such as network addressing, session management and positioning. In addition, it stores a description of all services provided by the devices in its area of coverage.

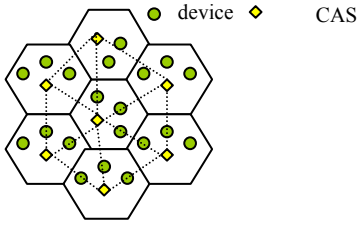

Figure 1. CAS distribution: each CAS manages the associate devices

MDM'05 , 5, 2005, Ayia Napa, Cyprus.

To address the scalability and mobility requirements of our environment, we introduce *communities* that group semantically related services. Communities correspond to an overlay network that connects such related services. Efficiency comes from the fact that communities provide a semantic index to similar services which may reside *anywhere* in the network.

2.1 Concept-based service queries

Which services belong to a particular community (i.e., which services are semantically similar) is built around the notion of a concept. *Concept* is a semantic notion. It is associated with a specific property which is described using a set of keywords, for example, "traveling", "weather" or "taxi reservation". To discover a service, a user specifies a set of appropriate keywords.

Besides concept-based service descriptions, in a mobile environment, it is critical to provide context-aware queries, that is, queries that match the current context of a moving user. Context-aware queries generalize location-aware queries that are queries that take into account only the user location. We model context as a set of (attribute, value) pairs. Context is used to limit the set of matching services returned to those that match also the specific context values.

In general, a context-aware query is a query that contains the results within the boundaries of a specific context.

Definition 1: *Context-Aware Queries (Qq) are composed from concept keywords and context pairs:*

Qq = <Concept{keywords},Context{(attribute, value)}>

where Concept keywords are used to identify the concept and (attribute, value) pairs are used to define the user context.

As an example, consider the case of a service for sports news. The concept in this example can be sports; the execution of the query "find me all services which provide sports news" should return all services which are related with the "sports news" concept, or a concept that is characterized by both these keywords. Depending on the query, the size of the result can be very large. Our goal is to contain the results by using the context information that characterizes the current environment of the user. For instance, in this example, if the user carries an iPAQ device, the discovered services should be suitable for it, that is, an appropriate context-aware query can be "find me all services which provide sport news displayable on an iPAQ PDA". This is expressed through the following context-aware query: Qq = <Concept{sports, news}, Context{(device, iPAQ)}>.

In addition to context-aware queries, we are also interested in keeping the result of a query up-to-date as the result set may change due to service and user mobility.

Definition 2: *Continuous context-aware queries (Qcq) are context-aware queries that notify the user for changes in the result set in a continuous fashion.*

Qcq=< Concept{keyword }, Frequency{types}, Context{(attribute, value)}>

Frequency types define the frequency by which the user should be alerted. Currently we support the "onFound" and "near by" frequency types.

An important issue is how to distribute the service directories so that both non-continuous and continuous context-aware queries are efficiently supported. CASs provide a level of distribution for the service directories, since each CAS maintains a local directory with the description of the services provided by the mobile devices in its coverage. However, services matching the concepts specified in the queries may be located in various CAS. To avoid the overhead of querying all CASs, communities are used. Communities are interconnected, creating a semantic overlay network that can be used for efficient service discovery.

2.2 Taxonomies for Organizing Communities

We need a way to classify and inter-relate communities. To this end, we use taxonomies whose elements are ontologies [5,6]. Such taxonomies take the form of a tree (see Figure 2). Each internal node of the tree corresponds to an ontology that describes a community (see Table 1). The node also refers to its children as well as to its parent node. Recursively this leads to a hierarchy of ontologies where each (deeper) level of the hierarchy provides a more refined and focused description of the concept. Each leaf of the taxonomy tree contains a subset of the description properties and functional attributes of the service's profile that belongs to the (parent) community. This profile summary is used to determine whether the service satisfies the query criteria and also to provide information for accessing the actual service (see Table 2).

Table 1. Community Ontology Properties

Community Name	The name of the community
Text Description	A brief description summarizing the concept of the community
Keywords Description	Keywords used to describe semantically the community
Parent	Reference to the parent community
Children	Reference to the children communities

Table 2. Summary Service Profile Properties

Service Name	The name of the service
Keywords Description	A keywords description summarizing semantically what service offers or what capabilities are being requested.
Provided By	A sub-property of role referring to the service provider
Geographic Radius	Geographical scope of the service, either at the global scale (e.g. e-commerce) or at a regional scale (e.g. pizza delivery)
Pointer	An abstract link to the full service ontology.
Community	Reference to the community

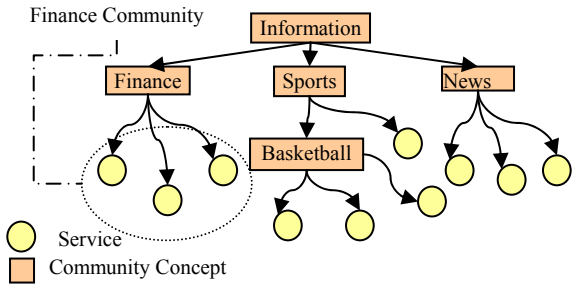

Figure 2. Global Taxonomy of Communities and Services

3. COMMUNITY OVERLAYS OVER CAS

To support the management of communities, we introduce the notion of a Community Administrator Server (CoAS). CoASs are

responsible for the creation and management of communities. Each CoAS maintains a community, which groups similar services provided by different CASs that can be located anywhere in the system. As the CoASs represent all communities, the complete taxonomy of the CoASs can be seen as an overlay network over the core system of CASs (see Figure 3). This overlay network instead of grouping services located in the same geographical domain, it groups services that are semantically related independently of their location. To create the overlay network of CoASs, each CAS propagates a summary of description ontologies of the services that it hosts (see Table 2) to the appropriate CoASs. Identifying the appropriate CoASs is achieved by using routing indexes based on Bloom filters described next.

As an initial global taxonomy tree, we use a basic classification of services taken from Google (e.g. a subset of Google's classification of urls). Using this classification we create the initial network of CoASs. Registering new services and matching is enhanced by using Wordnet [10], a lexical reference system that allows us to extract synonyms of each of the keywords used to express concepts. This is handled by the CAS when a service registers or a request for a service is submitted. We describe next the basic functionality provided by the CoAS.

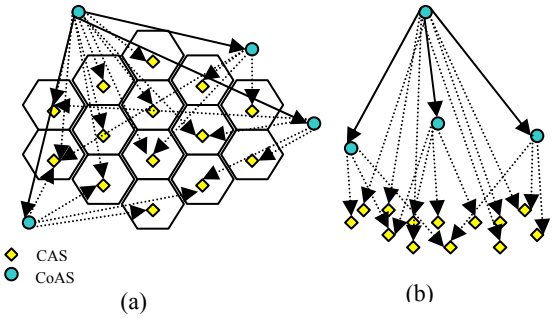

Figure 3. Distribution of CASs and CoASs (a) Geographical Distribution, (b) the Overlay Network of Communities

Service discovery: Querying for a service takes place when a CAS forwards a query to the CoAS that manages the community that serves the concept of the query. The CoAS is responsible for finding all services which satisfy the contextual condition posted in the query by the user.

New service registration: This operation takes place when a device registers its services to the CAS. The CAS stores locally the service ontology provided by the user, and propagates the service description to the communities (it could be more than one) which share the same concept with the service.

Service unavailability: A service provided by a device may become unavailable at any given time either voluntarily by its owner or because the device becomes unreachable due, for example, to network disconnections. In such cases, we do not delete the service from the community, but during service discovery we check for the actual service availability. The CAS is responsible to detect unavailability or availability and inform the appropriate CoAS. In case that a user already uses a service, which becomes unavailable, the user can extract the service community from the service profile, and use it to search directly to the community that groups similar services, bypassing the CASs.

Service update: An update operation at the community level takes place only when the semantic description of the service changes. In such cases, when the service profile changes, the CAS will propagate the changes only to the communities which store a summary of the service and only if this summary must be updated. Note that service mobility does not affect the community taxonomy. This is because location based queries are handled by the CASs, thus we do not have to update the communities whenever the device that owns the service changes location.

Using Bloom Filters to Locate a Community: To identify which CoAS match a given query, we use indexes based on Bloom filters. Bloom filters are compact data structures for probabilistic representation of a set that supports membership queries, that is queries on whether a given element belongs to a set. Bloom filters are used to determine which CoAS are relevant to a given query. At each CAS, there is one Bloom filter for each CoAS; we call this filter a community Bloom filter. The communities that match a service are the communities whose community Bloom Filters match all keywords describing the service.

In case there is an order among the keywords that follows the ontology schemas, we may use a query language (e.g., XPath-based [1]) that takes advantage of this order. To this end, we have introduced multi-level Bloom filters [2] that extend Bloom filters for supporting the efficient evaluation of path expressions including partial match and containment queries.

Continuous Context-Aware Queries: These types of queries are stored into the associated CoAS. Depending on the frequency condition, the CoAS will periodically push the results to the issuing device. There is an overhead for supporting continuous context-aware queries, since we need to check whether a new service matches any continuous queries registered at the corresponding CoAS. However, this overhead is small considering the overhead to provide this capability in the absence of the CoASs. In this case, all CAS would have to be checked whenever a new service is registered.

4. SERVING A QUERY VIA THE CoAS NETWORK

In this section, we present how the system supports continuous and non-continuous context-aware queries in more detail.

Context-aware Queries: The query execution steps are performed in collaboration between the CAS and the CoAS components.

1. A query for a service is formed by providing the concept keywords and submitted to the associated CAS. The order of the keywords corresponds to the concept hierarchy. The CAS enhances the query by appending the context keywords which define the user current environment. As an example, consider the following request: "*Find a service providing pop music clips for an iPAQ media player*". This request is formulated as follows: *Qcq=<Concept{music, pop, clip}, Context{(device, iPAQ)}>*. If the receiving CAS can satisfy the query, then it returns the results and the process terminates (location-based queries might be satisfied this way). If the request cannot be satisfied locally by the CAS, the CAS uses the Bloom Filters to identify which community (i.e., CoAS) serves the query

concept, in this example, the community "music clips". We assume that the community taxonomy contains such a community. In this case, the concept hierarchy is music | pop | clips; this hierarchy is used to direct the query to the appropriate community. In case that there is no community to serve the exact concept, we search for a community that serves the more general concept, in our example, this is "music pop". Upon finding the appropriate CoAS, the CAS forwards the query to it.

2. A CoAS upon receiving a query identifies all matching services. Matching is performed at a semantics level. The list of matching services includes all services that provide pop music clips currently registered in the CoAS unless other constraints are also imposed.

Continuous Context-Aware Queries: These types of queries differ from the previous ones in that they are stored into the CoAS. Depending on the frequency condition, the CoAS will periodically push the results to the issuing device. As an example, consider the following request: "*Find all services providing music clips for an iPAQ device and alert me when a new one is available"*. This request is formulated as follows:

<Concept{music, clip}, Frequency{ onFound}, Context{(device, iPAQ)}>.

1. The device submits the query to its current CAS. The CAS forwards the query to all appropriate CoASs, using the mechanism described earlier.
2. Each CoAS registers the query locally. Whenever a new service is registered to the CoAS, the CoAS checks whether there is any continuous query whose conditions may be satisfied by the new service. If this is the case, the query results are updated and the issuing device is notified.

5. PERFORMANCE EVALUATION

The core system infrastructure is composed from a set of CASs. These CASs are distributed across the network and independently manage the devices under their area of coverage. The CAS interface includes methods for (i) registering a new service, (ii) locating a service and (iii) retrieving context information (e.g. location). We implemented the Community Administrator Servers (CoASs) on top of the core system infrastructure [8]. CoASs are also distributed across the network. The interface provided by a CoAS consists of the following methods: (i) register a new service, (ii) locate a service, and (iii) get the results of a continuous query. The components that comprise a CoAS are:

1. *Service Ontology Directory:* lists all the service ontologies summaries currently handled by the specific CoAS.
2. *CoAS directory:* lists children CoASs.
3. *Query executor:* is responsible for matching an incoming query with service ontologies.
4. *Concept Alerts Directory*: is used to support continuous queries by providing triggers.
5. *CAS Directory*: lists all the CAS of the system.

We compare the communities approach with a centralized and flooding search approach. With flooding, each site searches locally for a service satisfying the query, if no service is found locally; the query is forwarded to the next site. The algorithm we use for the communities approach is: locate the best matching community (via bloom filters) and forward the query to it. We consider the total time that is required for a wireless client to send the request via a web-browser until getting the response in relation to the number of registered mobile services, the frequency and the number of arriving requests.

For the centralized and the flooding approaches, we only install the CASs (one at each machine) and adapt the searching mechanism accordingly, i.e., one service directory at a central location and one service directory at each site respectively. For the communities approach, the nodes of the CoAS taxonomy tree are stored randomly on the various nodes. The testbed configuration consists of 20 Pentium 4 2.000GHz workstations with 512 MB RAM running MS Windows 2000. This cluster remains the same for all tests. The total number of mobile services registered in the system is 20000. For the communities approach, the taxonomy tree is created randomly based on the keywords that characterize the 20000 registered services.

We perform two types of tests; the first one aims at testing the response type of the system under a somewhat normal load. We run a number of tests and for each test increment the number of service requests from 1 to 301 using an incremental step of 10 (i.e., we run 31 tests). Each test is repeated a number of times and the average response times are reported. In the second test, we test the scalability of the various approaches by performing a stress type of test. For each test, we increment the number of service requests from 1 per second to 301 per second using an incremental step of 5. That is, each test requires 60 seconds to submit all 301 requests. Again we perform a number of runs and the average times are reported. Each request is created randomly and again randomly submitted to a CAS.

5.1 Normal load

For the flooding approach, we performed three different tests. In the first test, every request is satisfied by the local CAS while in the other tests 25% and 50% of the requests respectively must be forward to the other CASs. In each such test, we assume on average a maximum of 3 hops per service request. For the communities approach, we test the system with taxonomy trees of 10 and 20 communities.

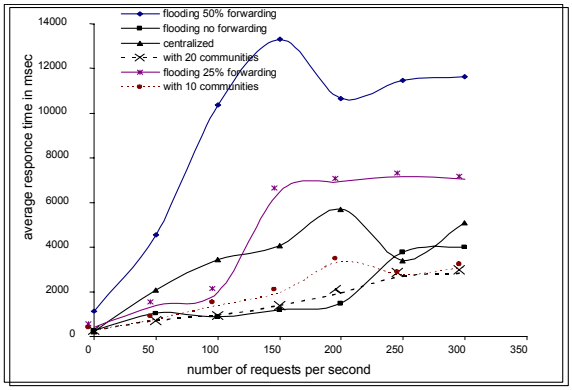

Figure 4. Comparison of centralized, flooding and communities

Figure 4 illustrates the performance results of the centralized, flooding and communities approaches. For the flooding approaches, the average response time to execute a query (i.e., locate a service) is quite large, especially when the number of requests is over 50. The best performance is depicted by the

"communities" approach. In fact, with both 10 and 20 communities, for a large number of requests, it performs better than the "flooding without forwarding", which is the optimal case for flooding, because the overall number of requests is distributed to a large number of community servers which have a small number of services to match.

5.2 Scalability test

In this test, we compare the performance of the three approaches over an over-loaded configuration. Again, we run a number of tests. In each test, we increment the frequency of the arrival of service discovery requests starting from 1 per second and incrementing it by 5 until 301 requests per second are reached. We keep track of the second at which each request is submitted and measure the average response time for each "second" cluster.

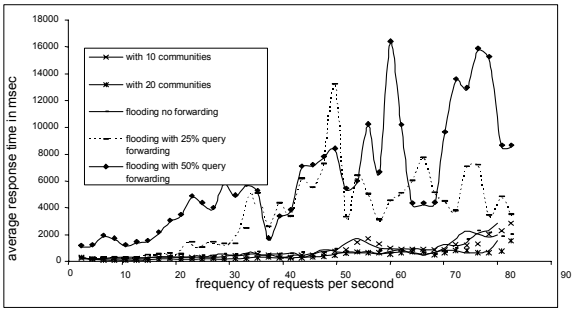

Figure 5. Comparison of centralized, flooding and communities

In Figure 5, we limit the rate up to 90, for clarity. Each CAS server allows a maximum number of 75 open connections, which is rather common in web servers. Even for relatively high frequencies, both the flooding without forwarding and the community approaches perform rather well. The flooding with forwarding approaches, however, reach very soon the allowed number of maximum connections, creating a significant congestion to the system and resulting in high response times.

6. RELATED WORK

Various service discovery mechanisms have been proposed in the literature. GloServ [1] uses a hierarchical schema similar to DNS to classify the registered services. The SLM system [3] adopts a distributed hierarchical tree structure to organize the SLM servers which may physically be located in wide-area networks. The GSD protocol [2] is based on the concept of peer-to-peer caching of service advertisements and group-based forwarding of service requests. The main difference of the above systems with our approach is that our work is based on a two-layer architecture: an overlay network on top of local servers. The SCAM [9] context model is based on an ontology which provides a vocabulary for representing and sharing context knowledge in a pervasive computing domain,, however, SCAM uses a centralized architecture.

7. CONCLUSION AND FUTURE WORK

In this paper, we propose creating a dynamic overlay network of related services where similar services form a community. In particular, each community is a set of pointers to semantically or contextually related services (for example, a community of weather services). Communities are organized in a global taxonomy whose nodes are related contextually. This taxonomy can be seen as an expandable, flexible and distributed semantic index over the core system, which aims at improving service discovery. We also presented a distributed service discovery mechanism that utilizes these communities for context-based service discovery. Our performance results indicate that service discovery works well under various workloads. As future work, we plan to explore the effectiveness of a query language for managing context. We also plan to study load-balancing by relocating communities close to their most frequent requestors.

ACKNOWLEDGMENTS: Work supported in part by the IST programme of the European Commission FET under the IST-2001-32645 DBGlobe project, IST-2001-32645 and a Bilateral research agreement between Cyprus and Greece (SemaNet project).

8. REFERENCES

[1] Arabshian, K., Schulzrinne H. GloServ: Global Service Discovery Architecture, *In Proceedings of the 1st First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous'04),* Boston, USA, 2004.

[2] Chakraborty, D., Joshi, A., et. Al. GSD: A Novel Group-based Service Discovery Protocol for MANETS, *In Proceedings of International Workshop on Mobile and Wireless Communications Networks* , Sweden, 2002.

[3] Gu, T., Qian, H. C. ,Yao, J. K. An Architecture for Flexible Service Discovery in OCTOPUS", *In Proceedings of the 12th International Conference on Computer Communications and Networks (ICCCN)*, Dallas, Texas, 2003.

[4] Koloniari, G., Pitoura, E. Content-Based Routing of Path Queries in Peer-to-Peer Systems. EDBT 2004: 29-47.

[5] Levy, A., Srivastava, D., Kirk., T. Data model and query evaluation in global information systems. *In Proceedings of the Intelligent Information Systems,* 5(2), September 1996.

[6] Ouzzani, M., Benatallah, B., Bouguettaya, A. Ontological Approach for Information Discovery in Internet Databases. *Distributed and Parallel Databases Journal*, 8:367-392, 2000.

[7] Pitoura, E., Abiteboul, S., Pfoser, D., Samaras, G., Vazirgiannis, M., et. al. DBGlobe: a Service-Oriented P2P System for Global Computing*, SIGMOD Record* 32(3): 77-82 (2003).

[8] Skouteli, C., Panagiotoy, C., Samaras, G., Pitoura, E. Communities: Creating and Quering Ad-hoc Databases based on Concepts*, In Proceedings of the International Workshop on Global Computing*, Italy, 2004

[9] Tao G., Xiao H. W., Hung K., P., Da, Q., Z. A Middleware for Context-Aware Mobile Services, *In Proceedings of the IEEE Vehicular Technology Conference (VTC Spring 2004)*, May 2004, Milan, Italy.

[10] Word Net: http://www.cogsci.princeton.edu/~wn/

[11] XML Path Language (XPath). World Wide Web Consortium, http://www.w3.org/TR/xpath