Communities: Concept-Based Querying for Mobile Services

Chara Skouteli[1], Christoforos Panayiotou[1], George Samaras[1], and Evaggelia Pitoura[2]

[1] Department of Computer Science, University of Cyprus,
CY-1678 Nicosia, Cyprus
{chara, cs95gp1, cssamara}@cs.ucy.ac.cy
[2] 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 is termed a community. Communities are organized in a global taxonomy whose nodes are related contextually. The taxonomy can be seen as an expandable, flexible and distributed semantic index over the system, which aims at improving service discovery. We present a distributed service discovery mechanism that utilizes communities for context-based service discovery. To demonstrate the viability of our approach, we have implemented an infrastructure for supporting communities as well as a prototype application that utilizes communities.

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 interconnected, 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. Our goal in the DBGlobe project is to provide both the theoretical foundations and the system infrastructure for effectively querying this database [21].

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 in services [14]. In this paper, we focus on the fundamental issue of how to efficiently query for services in such a global database. Service discovery in this dynamic environment, where providers and requestors are mobile, is more exigent than in the classic mobile environment where only the requestors can change location. Furthermore, the huge number of available mobile services demands an efficient service discovery mechanism.

We propose creating a dynamic overlay network above the core system to group together semantically related services, effectively creating a network of communities.

C. Priami and P. Quaglia (Eds.): GC 2004, LNCS 3267, pp. 59–76, 2004.

Each of these communities is a set of pointers to semantically or contextually related services that are distributed over the global mobile environment (for example, a community of weather services, or a community of services provided by PDAs). Communities are distributed and are effectively 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 decreasing the cost of service discovery. Providing flexible service discovery over communities allows us to expand the notion of context beyond the usual concept of location. In our work, a user's context is a set of mobile services belonging to a number of different communities. Having the communities managing concept-related services provides for a more efficient service discovery.

In a nutshell, in this paper, we propose: (i) a semantic grouping of mobile services over the global computational net, effectively creating a network of communities, and (ii) a distributed service discovery mechanism that utilizes these communities for context-based service discovery. We also study two types of context-based queries, containment and continuous queries that are central in this context. To demonstrate the viability of our approach, we have implemented the infrastructure for supporting communities as well as a prototype application that utilizes this infrastructure.

The remainder of this paper is organized as follows. Section 2 gives an overview of the core system architecture, while Section 3 describes mobile service directory in terms of communities and presents the types of queries we support. Section 4 describes the taxonomy and community architecture as an overlay network. Section 5 provides examples of query execution using communities. Section 6 presents our prototype implementation. Section 7 discusses related work and finally, Section 8 presents conclusions and future work.

2 Core System Architecture

DBGlobe is a global data and service management system [21]. It connects a number of autonomous devices and provides support for describing, indexing and querying their data and services (Fig. 1). DBGlobe employs a service-oriented approach in that data are wrapped as services. The mobile devices at the perimeter of the architecture are called Primary Mobile Objects (PMOs). They may function as service providers (servers), service requestors (clients) or both. They connect to the DBGlobe system and possibly directly to each other to exchange data through services. They register by providing appropriate metadata information depending on their role. Their number and location may change over time, as new PMOs enter or leave the system and existing PMOs relocate.

Besides these "walking" miniature databases of PMOs, DBGlobe system components, dispersed throughout the stationary network, store metadata information about PMOs, users and services, provide index and directory information, and query processing capabilities. These components are called Cell Administration Server (CAS). CASs also provide low-level functionality, such as network connectivity and mobility support.

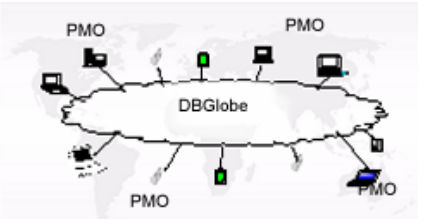


Fig. 1. The DBGlobe Layer

2.1 Primary Mobile Objects (PMOs)

A Primary Mobile Object (PMO) is any autonomous, electronic device capable of communicating independently with the CAS via some communication channel. The basic functionality of a PMO includes the ability to (a) request and retrieve data, (b) produce and share data, (c) create and publish a service and (d) communicate with a DBGlobe server and function as a source. We assume that every PMO has built-in a globally unique identity (like Ethernet adapter addresses or IMEIs in GSM phones) and possibly incorporates components that can capture context (such as GPS receivers, digital compasses and temperature sensors). In addition, it may host an application server (e.g, a web server) for executing services.

2.2 Cell Administration Servers (CAS)

The Cell Administration Servers (CASs) provide the basic DBGlobe functionality, including: (a) connectivity and addressing scheme, (b) service publication, (c) context determination support, (d) mobility support, (e) service life cycle tracking, and (f) service discovery.

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 (Fig. 2). A network of CASs constitutes the backbone that makes it possible for the PMOs to communicate and share data and services with each other. The CASs are interconnected through a network, e.g. the Internet. Although they can function autonomously, they are also aware of their neighbors (that manage geographically adjacent cells) and cooperate to increase the range of requests. In our current design [3], each cell represents the area of coverage of s network access point. We assume that every PMO (including stationary devices) is associated with at most one cell at any given time (e.g., by keeping a live connection to the cell's defining network access point).

Each CAS can independently manage the PMOs which enter its area of authority. It keeps track of the PMOs that enter or leave the cell's boundaries. It stores metadata describing each PMO, the context and the resources offered and assists the user to locate services by semantically matching requests with existing service descriptions. It also provides basic services to visiting PMOs such as network addressing, session management and positioning. Each cell can support large numbers of PMOs moving

inside its area and acting as sources or requestors of information. The CAS module consists of:

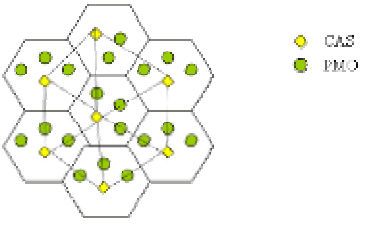


Fig. 2. The System Geographical Distribution CAS manage the covered PMOs

1. A service directory that lists all the services offered by PMOs in the cell.
2. A service description repository of the local services
3. A CAS directory, containing addresses of other CASs.
4. A community directory, containing addresses of the available communities
5. A location management sub-system that keep tracks of the location of the registered to that cell PMOs
6. A device type and a PMO repository containing the list of device types and PMOs available in the cell and their profiles,
7. A temporal profile manager for storing the connection times of devices, discovering patterns and estimating probabilities of next appearance. A server can also keep historical data and compute statistics about their mobility habits to assist proactive behavior.
8. A service discovery mechanism for locally residing services.

The distributed nature of the system, however, requires an efficient distributed service discovery mechanism which is maintained collaboratively among the various CASs. The basic idea is to use a global service director that utilizes a distributed hierarchical service taxonomy structure to assist the user to locate services by semantically matching requests with existing service descriptions. This is achieved using service communities described next.

3 Service Discovery Based on Concepts in Mobile Environment

Most common service discovery approaches are based on service registries that match service requests with available service descriptions [19]. However, in a global computing environment, where (i) service providers and requestors are mobile and (ii) service unavailability (due, for example, to wireless disconnections) occurs frequently, there is a need for more sophisticated service discovery mechanisms. For instance, users may want the results of service discovery to be adapted to their current state and be updated in a continuous fashion, e.g. if a user is driving, she is not able to type, but still wishes to find with minimum efforts results which are useful to her at the current time. To achieve this, we need an infrastructure that can collect and manage context information about the various system entities.

Context information is defined as: any information that can be used to characterize the situation of an entity, where an entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and the application themselves [17]. In the mobile environment, the most common context information includes location, user preferences, user situation and device characteristics. This information can be used to filter the results and provide more accurate and useful services. However, context information alone is not able to minimize the searching domain; in the case of a distributed system such as DBGlobe, we need to query and apply the user's context on all available service directories in each CAS. To avoid this, we use the notions of concept and communities, which allow us to contain any query within a concept efficiently and in a distributed manner.

Communities group semantically related services that are distributed over the network. 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 and describes a specific property, for example, "traveling", "weather" or "taxi reservation". Each concept is described through a set of keywords. Then, given a set of appropriate keywords, the matching concept or concepts (and thus communities), are identified and the appropriate services are selected. Having the universe of services divided and grouped into concepts allows a distributed and efficient implementation. Thus, if we wish to find a "weather" service for Cyprus, we just forward the query to the "weather" community and filter the results by using the location context.

3.1 Context Aware Queries

In a mobile environment, where users are moving, it is critical to provide context aware queries that (a) give concept information in a contained fashion and (b) provide results in a continuous fashion. Contained queries aim at containing the results usually within location boundaries (e.g., location-based queries). In our environment, however, the results are contained around a more general context not just location. Continuous queries are always active and aim to inform the user whenever the conditions posted in a query are satisfied (e.g. "Find services which provide videos from Greece and alert me when a new one appears").

Definition 1: A Concept Containment Query is a query that contains the results within the boundaries of a specific contextual concept. The services that are queried are related to a specific concept. Concept Containment Queries (Qccq) are composed from concept keywords and context pairs:

$$Qccq = \langle Concept\{keywords\}, Context\{(attribute, value)\}\rangle$$

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 query can be "find me all services which provide sport news displayable on an iPAQ PDA". This is expressed through the following query: Qccq = <Concept{sports, news}, Context{(device, iPAQ)}>.

In addition to containment queries, we are also interested in keeping the result of a query updated as the result set might change due to service and user mobility. This is essential since services are mobile and dynamic as PMOs may move, join or leave the system. Continuous queries aim at keeping the results of a query up-to-date. The ability to satisfy concept containment continuous queries in a global mobile environment is critical.

Definition 2: Concept containment continuous queries (Qcccq) are containment queries that notify the user for changes in the result set in a continuous fashion.

Qcccq=< 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" type. An example concept containment continuous query is: "Find close by services that provide photos of Greece displayable on an iPAQ and alert me whenever a new one becomes available". This is expressed as <Concept{Greece, photos}, Frequency{onFound}, Context{(location, "current location"), (device, iPAQ)}>. This query alerts the user whenever a service which provides photos displayable on an iPAQ is near by.

Assuming that concepts are organized in some order, the order of keywords can direct query processing. Keywords following a concept ordering may improve service discovery. For example, the query "find me all services which provide sports news displayable on an iPAQ PDA" should return all services which are related with the "sports news" concept where the concept "sports news " is a sub-concept of sports. An order-preserving query language could indicate this as follows: sports | news. Other constructs such as "or", "and" could also be defined. Thus, the query above would be expressed as: <Concept{sports | news}, Context{(device, iPAQ)}>.

An important issue is how to distribute the service directories so that both containment and continuous 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 (PMOs) in its coverage. By doing so, we are able to efficiently support location-based queries, that is, queries with a single location attribute. However, it is not possible to efficiently support more general concept queries, since a matching service may be registered at any CAS directory and thus all of them need to be queried. To avoid this overhead, we need a mechanism that will group services in multiple ways not only based on their location. To this end, we propose a query mechanism based on virtual directories, called communities, that cluster similar services. Communities are interconnected, creating a semantic overlay network that can be used for efficient service discovery.

3.2 Organizing Communities Using Taxonomies

Just having a collection of communities does not entirely solve the problem. We also need to organize the communities. To do so, we need a way to classify and inter-relate communities. We assume that communities are described through ontologies. Ontologies are used to fully describe entities [6, 7, 8, 10, 20], in our case, communities and services. In order to express such classifications and interrelations, we use taxonomies whose elements are the aforementioned ontologies.

Table 1. Community Ontology Properties

communityName	The name of the community
textDescription	A brief description summarizing the concept of the community
keywordsDescription	Keywords used to describe semantically the community
Parent	Reference to the parent community
Children	Reference to the children communities

Such taxonomies take the form of a tree (Fig. 3). Each internal node of the tree corresponds to an ontology that describes a community (Table 1). The node also refers to its children which are under its contextual umbrella as well as 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 [20] that belongs to the (parent) community. This profile summary can be used to determine whether the service satisfies the query criteria and also to provide information for accessing the actual service (Table 2).

Given a taxonomy (i.e. a tree structure of concepts) we can run an inquiry for a specific topic by performing a top down search for matching ontologies within the taxonomy tree. During search, the parent ontologies are used to narrow down the contextual domain of the children nodes. In this way, we only get related services. For instance, we avoid asking for Asian culture services and getting hotel reservation services; we always get services from the right concept.

Using communities allows queries to run faster. The efficiency comes from the fact that communities are a collection of pointers to services belonging to a particular concept which may reside anywhere in the network. The question here is why not using one centralized unified index instead of communities. The first reason is scalability. The other one stems from the heterogeneity of the environment. A third factor is the semantic nature of the required index. We need an index that can give as the location of a service based on its semantic description, thus complicating the structure of the index. Finally, as the complexity of the index increases the cost of updating it becomes prohibiting. Communities, which are in essence a semantic index, tackle all these problems:

Table 2. Summary Service Profile Properties

serviceName	The name of the service
keywordsDescription	A keywords description summarizing semantically what service offers or what capabilities are being requested.
providedBy	A sub-property of role referring to the service provider
geographicRadius	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.

- Heterogeneity in the environment does not affect communities as long as we use a standardized way of describing the available resources/data.
- Communities provide semantic based indexing of services.
- Updating a community (which has much fewer members than a unified index) is more cost effective. The updates are distributed to a number of communities, a fact that limits the load on each individual community.
- Communities can relocate as needed, thus providing load balancing.
- The CASs infrastructure provides a local index. That is, it is able to efficiently support context aware queries based on location, because the notion of location restricts the number of CASs that we have to contact.
- The ability to distribute communities provides scalability.

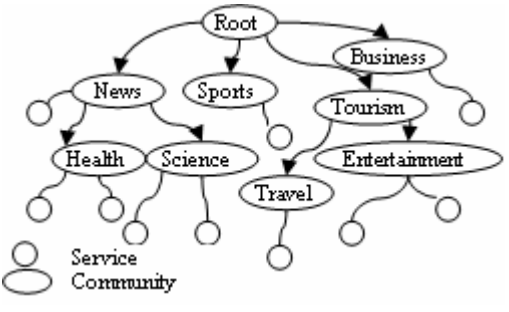


Fig. 3. Global Taxonomy of Communities and Services

Organizing the service directory around concepts and communities allows us to efficiently distribute the directory (i.e., distribute the communities) over the network.

4 Communities as an Overlay Network Over CAS

To implement 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, and it 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 (Fig. 4). This overlay network instead of grouping services located in the same geographical domain, it groups services which are semantically related independently of their location. To create the overlay network of CoAs, 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 [4] described in Section 4.2. The complete overlay network of CoASs constitutes a global distributed taxonomy tree of communities. Figure 4 shows a possible configuration and distribution of such a network.

4.1 Managing the CoAS Taxonomy Tree

The CoAS topology is a hierarchy of ontologies. For managing this distributed topology, important operations include (i) updating its content when a new service is registered to the system or a PMO disconnects and thus its services become unavailable and (ii) load-balancing the taxonomy when a node/community becomes overloaded with service descriptions.

Construction: 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.

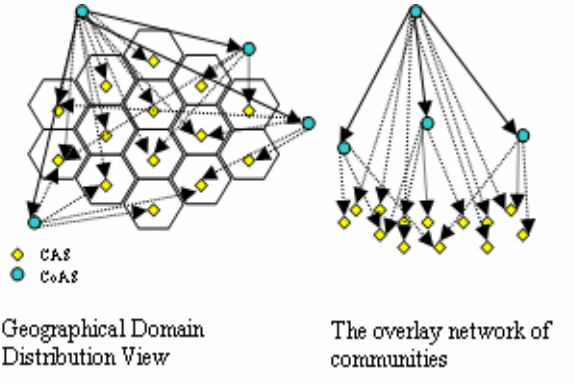


Fig. 4. Distribution of CASs and CoASs

New Service Registration: This operation takes place when a PMO 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. Utilizing Bloom filters [4, 5] in combination with the services' descriptions and the community concepts allow the CAS to identify the appropriate CoASs. Deletion of a service is handled in a similar manner.

Service Unavailability: A service provided by a PMO may become unavailable at any given time either voluntarily by its owner or because the PMO becomes unreachable due, for example, to network disconnections. In such cases, we do not delete the service, so during service discovery we check for the actual service availability. The CAS is responsible to detect unavailability or availability and inform the appropriate CoAS.

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 PMO that owns the service changes location.

Balancing Communities: In case where a community becomes too large, reducing its efficiency, it can be split into two sub-communities. To decide which concept should be used to build the new community, we cluster the existing community and select the concept of the largest cluster. Two new CoASs are created to manage the new sub-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 to find all services which satisfy the contextual condition posted with the query. In Section 5, we present the query mechanism in more detail.

4.2 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. A Bloom filter BF of size m is a vector of m bits. Initially, all m bits are set to 0. Consider a set A = {a1, a2, …, an} of n elements. A number of k independent hash functions, h1, h2, …, hk, each with range 1 to m are used as follows. For each element $a \in A$, the bits at positions h1(a), h2(a), ..., hk(a) of BF are set to 1. Note that a particular bit may be set to 1 many times. Given a query for an element b, we check the bits at positions h1(b), h2(b), ..., hk(b). If any of them is 0, then certainly b is not in the set A. Otherwise we conjecture that b is in the set, although there is a probability that this is not the case. This is called a false positive. Parameters k and m can be chosen so that the probability of a false positive is acceptable.

Bloom filters are used to determine which CoAS should be updated when a new service is added, deleted or becomes unavailable from the system. They are also used to find which CoASs are relevant to a given query. In particular, given a service description, using Bloom filters, we can efficiently locate the appropriate CoASs.

At each CAS, there is one Bloom filter for each CoAS; we call this filter a community Bloom filter. Let CBF(A) be the community Bloom filter that corresponds to CoAS A. To construct the CBF(A), the k hash functions are applied to all concepts (keywords) that describe community A, and the associated bits of the filter are set to 1. Given a service s, to find the CoASs that match the service s, we apply the hash functions to each of the keywords that describe the service. For each such keyword of the service, we apply the hash functions and check which community Bloom filters match it. A filter matches the service, if all bits at the corresponding positions are set to 1. The communities that match the service s are the communities whose community Bloom Filters match all keywords describing the service.

In case that there is order among the keywords that follows the ontology schemas, we may use a query language that takes advantage of this order. This should most likely be a query language based on XPath [3] that allows us to exploit the structure of the schemas as well as their content. To this end, we have introduced multi-level Bloom filters [5] that extend Bloom filters for supporting the efficient evaluation of path expressions including partial match and containment queries. Multi-level Bloom filters are used to represent the CoAS taxonomy. In particular, instead of maintaining a simple Bloom filter for each CoAS, we maintain a multi-level one.

5 Servicing a Query Via the CoAS Network

In this section, we present how the system supports concept and continuous containment queries.

5.1 Concept Containment Queries

The query execution steps are performed in collaboration between the CAS and the CoAS components.

1. A PMO, service or user compose a query by providing the concept keywords and submit the query to the associated CAS. The order of the keywords corresponds to the concept hierarchy. The CAS composes 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)}>

2. If the receiving CAS can satisfy the query then it returns the results to the issuing PMO, service or user and the process terminates (location-based queries might be satisfied this way). If the request can not 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 better 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 "music pop" and the keyword "clips" is submitted as a context constraint in the query. Upon finding the appropriate CoAS, the CAS forwards the query to it. If there is no community to serve the concept, the request is forwarded to the root community for a top down search.

3. A CoAS upon receiving a query identifies all matching services. Matching is performed at a semantics level. All matching services are reported in a list. For the example query, the resulted list will contain all services which provide pop music clips currently registered in the CoAS unless other constraints are also imposed.

Example Query 1**:** *"Find all services providing photographs of Parthenon"*. Assuming that there is no community to serve the concept "Photographs of Parthenon" the keyword Parthenon becomes a context keyword and is used to filter the services which are registered to the photograph community. To this end, we use a reserved attribute name, called "concept". This query is formulated as follows:
Q1 = <Concept{photograph}, Context(concept, Parthenon)}>

5.2 Concept Containment Continuous Queries

These types of queries differ from the previous ones in that they must be stored into the CoAS. Depending on the frequency condition, the CoAS will periodically push the results to the issuing PMO. To better understand this mechanism, consider the following request: "Give me 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{new, onFound}, Context{(device, iPAQ)}>.

The PMO submits the query to its current CAS. The CAS forwards the query to all appropriate CoASs, using the mechanism described earlier.

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 PMO is notified.

There is an overhead for supporting concept containment continuous queries, since we need to check whether a new service match 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.

Example Query 2: *"Give me services providing photographs of Parthenon and alert me when a new one is available"*. This query is expressed as follows:

Q2 = <Concept{photograph}, Frequency{new, onFound}, Context{(concept, Parthenon}>

Example Query 3: *"Give me services providing finance services and alert me when a new one is submitted by the user "xak"*. This query must be forwarded to community managing the concept "Finance" and with context the service provider. This is expressed as:

Q3 = <Concept{finance}, Frequency{new, onFound}, Context{(provider, "xak")}>

6 Implementation and Prototype

The core system infrastructure is composed by a set of CASs. These CASs are distributed across the network and independently manage the PMOs under their area of coverage. Low level communication between the available CASs is achieved by using RMI. For extensibility, we also manipulate CASs as web services. 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. CoASs are also distributed across the network and can be manipulated as web services. The main system components which have direct access to the CoASs are the CASs. Communication of these components is achieved either with RMI or web services technology. The interface provide 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.

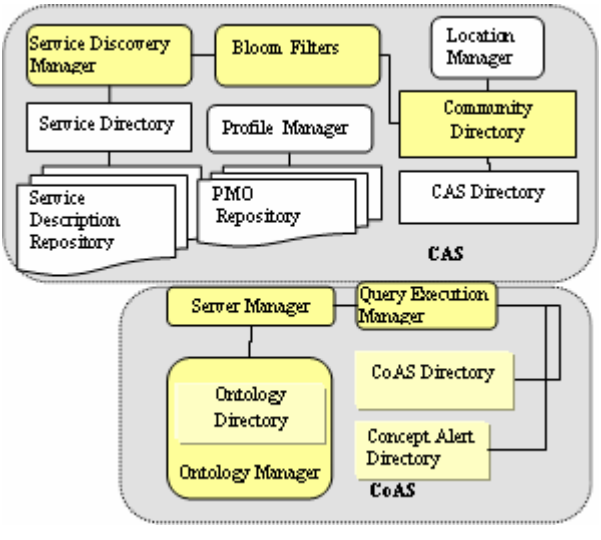


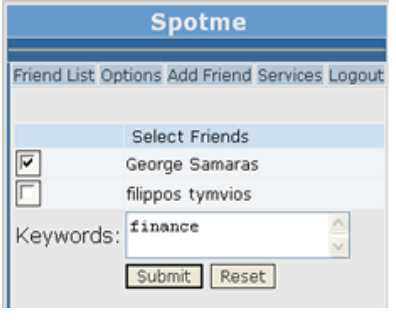
Fig. 5. The CAS and CoAS Server Architecture

6.1 Community Administrator Server (CoAS) Architecture

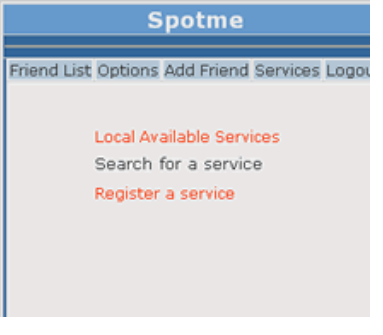
The components that comprise a CoAS are the following (Fig. 5):
Service Ontology Directory: lists all the service ontologies summaries currently handled by the specific CoAS.

1. CoAS directory: lists children CoAS.
2. Query executor: the most important component of the CoAS, as it is responsible for matching an incoming query's criteria with service describing ontologies. In effect, it is the context awareness query processor.
3. Concept Alerts Directory: used to better support continuity for incoming queries by providing triggers for them.
4. CAS Directory: lists all the CAS of the system.

a. Clips of SpotMe Application Running on a Sony Clie : Colleagues displaying



b. Creation of a Concept Containment Continuous Query



c. View the Results of the Alert

Fig. 6. SpotMe Prototype

6.2 SpotMe: A Context Aware Application

One of the main objectives of our infrastructure is to support the development of context aware applications. To demonstrate the capabilities of our system, especially distributed service discovery, we implemented a context aware prototype application called SpotMe on top of the CAS and CoAS infrastructure. The goal of the application is to create a collaborative environment where groups of users connect to the system to share their services. The prototype application is web-based and supports both continuous and containment queries.

Figure 6 exhibits some of the application capabilities. Fig. 6.a shows the basic screen of the application where a user has organized her friends into groups. As shown in Fig. 6.c, the user is able to view the local available services; this option exhibits the infrastructure capabilities to provide location-based queries. Moreover, the user has the option to register her services and search for services. Figure 6.b details the search capability where the user selects a list of friends and also submits a set of keywords which define the concept of the services. A possible concept containment continuous query example is "Inform me when one of my friends submits a new financial service" where the concept of this query is "financial" and the context is the selected list of users. To demonstrate concept containment continuous queries, the application offers the option to users to be alerted whenever one of their selected friends registers a related service. Thus, the search in Fig. 6.b is translated to the query:

<Concept{financial}, Frequency{new, onFound}, Context{(users, friends)}.

The following environment has been used to test the prototype implementation: a CAS network consisting of three CAS interconnected through the internet. All of them are also internet gateways, two of them allowing near-by users to access the network via wifi and one of them via Bluetooth. The initial overlay network of communities, shown in Fig. 7, is also interconnected through the internet. We used the following mobile devices: a Sony Clie, a Toshiba and an iPAQ connected via wifi and Bluetooth. Figure 6.a shows clips of the application's interface on a Sony clie.

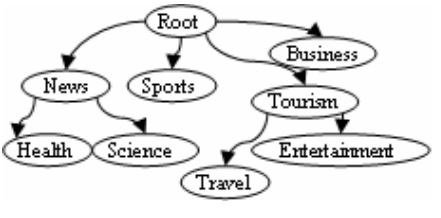


Fig. 7. Initial Community Hierarchy

7 Related Work

GloServ [12] is a service discovery system for a mobile environment that shares same common design issues with our work. More specifically, GloServ uses a hierarchical

schema to classify the registered services. Its architecture is similar to DNS in that it contains root name servers and authoritative name servers that manage information about services. GloServ classifies the hierarchy of services and establishes RDF schemas that describe each type of service. The SLM system [18] is another service discovery architecture that shares some ideas with our approach. The SLM service discovery system consists of SLM servers, services and SLM clients. An SLM server is a service information repository, providing SLM clients with access to all available services. SLM clients can search for services on behalf of end users. The system adopts a distributed hierarchical tree structure to organize SLM servers which may physically be located in wide-area networks. Both approaches create a hierarchy between the directory servers where the services are registered, while, in our approach the directory servers (CASs) are interconnect in a graph structure; but we provide the hierarchy of the available services on top of this graph structure. Our approach is more scalable because when a CAS does not respond, we are still able to find a service because all services are indexed by the taxonomy tree. Conversely in the case that a CoAS is not available, the query can be executed by the CAS.

The SCAM [13] context model is based on an ontology which provides a vocabulary for representing and sharing context knowledge in a pervasive computing domain, including machine-interpretable definitions of basic concepts in the domain and relations among them. To capture a great variety of context, they divide a pervasive computing domain into several sub-domains, e.g., home domain, office domain, vehicle domain, etc; and define individual low-level ontology in each domain. The separation of domains reduces the burden of context processing and makes context interpretation possible on mobile thin clients. The important difference with our approach and SCAM is that SCAM uses a centralized architecture, OSGi-compliant, mobile service gateway.

8 Conclusions and Future Work

In this paper, we consider semantic service discovery in a global computing environment. We described a low-level architecture of directory servers, each one of which maintains information about the services offered by the devices inside its area of coverage. We proposed creating a dynamic overlay network above this network of servers that groups semantically related services, effectively creating a network of communities. 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. To demonstrate the viability of our approach, we have implemented the infrastructure for supporting communities as well as a prototype application that utilizes this infrastructure. 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.

References

1. Samaras,G., Spyrou,K., Pitoura,E., Dikaiakos, M.: Tracker: A Universal Location Management System for Mobile Agents. Proc. The European Wireless 2002 Conference, Next Generation Wireless Networks: Technologies, Protocols, Services and Applications, Florence, Italy (2002) 572–580
2. Bray, T., Paoli , J., Sperberg-McQueen, C. M.: Extensible Markup Language (XML) 1.0 Specifications. World Wide Web Consortium, http://ww.w3.org/TR/Rec-xml
3. XML Path Language (XPath). World Wide Web Consortium, http://www.w3.org/TR/xpath
4. Koloniari, G., Pitoura, E.: Content-Based Routing of Path Queries in Peer-to-Peer Systems. EDBT Heraclio Greece (2004) 29–47
5. Koloniari,G., Pitoura, E.: Bloom-Filters for Hierarchical Data, Proceeding of the 5th Workshop on Distributed Data and Structures (WDAS) (2003)
6. Services Definition Language (WSDL), Web page, http://www.w3.org/TR/WSDL.
7. Ouzzani, M., Benatallah, B., Bouguettaya, A.: Ontological Approach for Information Discovery in Internet Databases. Distributed and Parallel Databases an International Journal, Volume 8, Issue 3 (2000) 367–392
8. Levy, A. , Srivastava, D., Kirk., T.: Data model and query evaluation in global information systems. Intelligent Information Systems, 5(2) (1996)
9. Lee,C., Helal, D.: Context Attributes: An Approach to Enable Context-awareness for Service Discovery. In the Proceedings of the 2003 Symposium on Applications and the Internet,(SAINT'03), Orlando, FL, USA, (2003)
10. Pfoser,D., Tryfona, N.,Verykios, V.: Services-Based Data Management in a Global Computing Environment. Workshops (WISEW'03) Roma (2003) 45-53
11. XML Query (XQuery). World Wide Web Consortium, http://www.w3.org/XML/Query
12. Arabshian ,K., Schulzrinne, H.: GloServ: Global Service Discovery Architecture, Department of Computer Science, Columbia University, New York (2004)
13. Tao Gu , Xiao Hang Wang , Hung Keng Pung , Da Qing Zhang : A Middleware for Context-Aware Mobile Services, IEEE Vehicular Technology Conference (VTC Spring 2004), Milan, Italy (2004)
14. http://www.w3.org/2002/ws/
15. http://www.w3.org/2001/04/30-tbl
16. Pitoura,E., Samaras, G., :Locating Objects in Mobile Computing. IEEE Transactions on Knowledge and Data Engineering Journal (TKDE). Vol. 13, No. 4 (2001) 571–592
17. Dey, A.K. , Abowd, G.D.: Towards a Better Understaning Context and Context-Awareness. In the Workshop on The What, Who, Where, When, and How of Context-Awareness, The Hague, The Netherlands (2000)
18. Gu,T., Qian, H. C. ,Yao, J. K., Pung, H. K. :An Architecture for Flexible Service Discovery in OCTOPUS", Proc. of the 12th International Conference on Computer Communications and Networks (ICCCN), Dallas, Texas (2003)
19. UDDI: The UDDI Technical White Paper. http://www.uddi.org

20. DAML-S Coalition:DAML-S Service Description for the Semantic Web, In Proceedings of The First International Semantic Web Conference (ISWC) Sardinia, Italia (2002)
21. 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) (2003) 77–82