A Multi-tier Ubiquitous Service Discovery Protocol for Mobile Clients

ABSTRACT

Service discovery and delivery problems have recently been drawing much attention from researchers and practitioners. SLP, Jini, and UPnP are among the few emerging service discovery protocols. Although they seem to provide a good solution to the problem, there is an unaddressed need of more sophisticated location and context-aware service selection, and mobile device support. In this paper, we introduce a multi-tiered mobile service discovery architecture that addresses the dynamics and the added requirements of mobility. We introduce the concept of *context attribute* as an effective, flexible means to exploit relevant context information during the service discovery process. Context attributes can express various context information including client device capability, service-specific selection logic, and network condition. We also present the Localized Service Propagation protocol (LSP) for mobile service discovery and advertisement. We describe our architecture, concepts and protocols, and present a performance scalability study of the LSP protocol.

KEYWORDS: *Mobile service discovery & advertisement, context attribute, context-aware service selection, service discovery performance.*

1. INTRODUCTION

Since *ubiquitous computing* was envisioned by Mark Weiser [29] in early 90s, it has been widely accepted as the ultimate technology for post-PC era. Although it has often been quoted as "pervasive computing", "invisible computing", or "anytime, anywhere, any device computing" depending on people and their orientations, all largely mean the same thing: *computation embedded in the environment is available everywhere to assist users in accomplishing their tasks.* Interactions in that ubiquitous world are envisioned to be modeled as service transactions. This includes service discovery, selection, leasing, and rendering. Interactions may also entail the composition of basic services into newer, value-added ones. To realize this vision, the infrastructure should be able to effectively support user mobility and a wide range of mobile devices inherent in the ubiquitous computing environment.

In the last decade, enormous research efforts have been concentrated on context-aware computing and it is now being considered as a key technology to cope with challenges brought forth by the ubiquitous computing vision. According to Dey *et al.*, the context is "any information that can be used to characterize the situation of an entity, where an entity can be a person, place, or physical or computational object" [9]. A good survey on how context can be beneficial in the ubiquitous environment is given by Chen *et al.* at [6].

As in the general context-aware computing research, context information is key to the ubiquitous service discovery problem. In other words, context-awareness enables the right services to be delivered to the right users (e.g., at the right time and place, in the form and delivery mode most appropriate to the mobile device, etc.) The users' intention is implicitly understood in the situations in a non-obtrusive manner by the effective interpretation of context. We envision the ubiquitous service discovery process to consist of four phases, captured by the four layers shown in Figure 1.

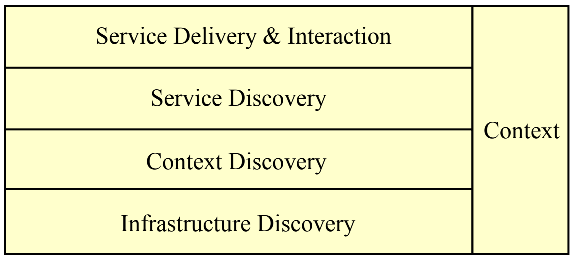


Figure 1- Ubiquitous Service discovery layers

The four phases are infrastructure discovery, context discovery, service discovery, and subsequent service delivery phase. The infrastructure discovery cares about lower-level resource discovery such as system or network resource acquisition: One example is IP address allocation through DHCP for mobile, visiting devices. Another example is Bluetooth SDP (Service Discovery Protocol) [3], which is designed to discover lower-level services, for instance, LAN access points within range. Although the context discovery and the service discovery are shown as logically separate layers, our view is that the two layers are closely coupled as the latter uses the service of the former. Client queries are resolved into references to services considering their contexts captured by the context discovery layer. Finally, a discovered service is delivered to the client in an appropriate form and mode with regard to the combined context of the service and clients. For instance, a light-weight version of a service can be selected for delivery into a less powerful device; or a thin client adaptor can be used to deliver a heavy-weight service to a resource poor device [36]).

As far as small, proximity-based service discovery is concerned, the context-awareness problem has been well addressed by several research prototypes [10,30,32], where individual phases are somewhat separately performed. The prototypes have demonstrated particular applications running in small, homogeneous environments, where much of the context is primarily derived from the proximity of service registries, service providers, and users. These prototypes, however, left other important aspects of ubiquitous service discovery unaddressed. Most critically, they provided no support for wide-area coverage and did not sufficiently exploit the use of context in dynamic service discovery. Although the wide-area coverage aspect is to some extent addressed in general service discovery frameworks [14,20,28], context is not sufficiently utilized. Recognizing this opportunity, we developed a novel approach to service discovery, in which we provide support for mobility (local and wide-area coverage) and context. Our approach will enable the most appropriate service to be discovered by the nomadic or mobile user, in her home network or foreign roaming network; on a desktop environment or off a mobile device. Before we present the details of our ubiquitous service discovery approach, we describe its distinguishing features.

First, the aforementioned four layers of service discovery process need to be formally established and tightly-coupled to better serve the users' needs of dynamic service discovery. Implicitly captured context information plays a pivotal role to glue these layers together and let them cooperate towards their final goal, which is successfully delivering the most appropriate service to the user's device.

Second, there already exist several service discovery frameworks that are diversified on grounds of different target devices, application domains, and underlying networks. Another reason of such diversity is the coverage limitation of each service discovery protocol, which requires client devices to understand multiple service discovery protocols. For example, one service discovery protocol is specialized in local services, while another is dedicated for global services. Resource-poor mobile devices can't afford to host and accommodate heterogeneous and fragmented service discovery protocols. Users must also be shielded from these disparate technologies via a single, integrated service discovery framework.

Third, although a few systems have already been proposed that support wide-area service discovery, their scalability are questionable. We propose a new scheme in which service advertisements attenuate, as they propagate throughout the network, and eventually cease beyond a certain range. In other words, we provide means to localize service discovery traffic to control the entire global service discovery traffic.

Fourth, there is little user mobility support in existing service discovery systems. As they move around, users should be able to see identical services on different devices at different places. Our architecture supports the service mobility via a user service profile concept.

Finally, going beyond local areas, the importance of the right service discovery gets magnified. It is because users will likely to experience much broader range of service quality in wider areas, depending on which service instances are selected. Current server selection research addresses this issue (e.g., the best service instance selection among identical service replicas). Service selection can be applied as a second step, occurring after the service discovery, in a service-specific channel separate from the general service discovery network. This means that by existing approaches, the users have to go through one more step, i.e., the server selection mechanism, to reduce candidate services discovered by service discovery protocols. These separate service discovery protocols and server selection procedures are incorporated into one framework in our proposed architecture.

The rest of the paper is organized as follows. Section 2 surveys related research such as server selection mechanisms and service discovery protocols. In section 3, we describe how and what context information can be used in the course of ubiquotous service discovery. Based on the context awareness, our service discovery architecture is presented in section 4. Finally, section 5 presents basic simulation results to compare our global service discovery scheme with existing protocols.

2. RELATED WORK

Our work on designing and developing a prototype for context-aware service discovery framework has been influenced by several emerging fields of researches and technologies. The key technologies that have motivated us to pursue this research are service discovery and subsequent delivery, server selection, and context-aware computing technology. The server selection problem is concerned about a particular type of services, e.g., selecting the best Web server among identical replicas. In contrast, the service discovery problem is to locate general, all possible types of services. One of major goals of our work is to augment service discovery framework by incorporating server selection schemes through context information.

2.1. Service Discovery

Most of service discovery protocols are based on the announce-listen model where periodic multicast is used for service announcement and discovery. Jini [19] is built on top of Java object and RMI system. A service proxy object is registered with a service registry, called a lookup service, which constitutes announcement process. A client downloads the service proxy through discovery process and invokes it to access the service. Services are identified by means of Java class hierarchy and attached service attributes.

The service registry is called a DA (Directory Agent) in IETF SLP [14,15]. Services are advertised and discovered through it between servers (Service Agents) and clients (User Agents). In all message exchanges among DAs, SAs, and UAs, services are described by *service: URLs* which are composed of a service type and a set of attribute-value pairs.

In UPnP [28], there's no such a central service registry. Services multicast their presence announcements periodically, while control points learn services of interest to them either by passively listening to those advertisements or by actively multicasting discovery messages searching for devices or services. The advertising message contains a few, essential specifics about the service and URLs for more detailed information. By following these URLs, a control point can retrieve XML descriptions that provide detailed information for subsequent 4 steps in UPnP networking: description, control, event, and presentation.

UPnP is designed to accommodate home network or small office network environments, whereas SLP and Jini can cover larger size network, i.e., enterprise networks. This limitation comes from UPnP's heavy use of multicast, which can be avoided by service registries in SLP and Jini. UPnP leverages IP and Web technologies, including TCP/IP, HTTP, and XML: Services are described in declarative XML language and communicated via HTTP. The absence of central service registry and adopting Web technologies make UPnP more suitable for peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs. The extreme of impromptu peer-to-peer connectivity support is found at MOCA [2] where each mobile device maintains its own service registry on it. Therefore, it enables ad-hoc collaboration on the spot without any help from the infrastructure, making it an ideal choice for dynamic, very small networks. On the opposite side of the coverage lies SLP. It achieves much wider coverage by minimizing the use of multicast through service registry. However, it can also operate in registry-less mode in case of a small network. Also, SLP is well aligned to other Internet technologies. For instance, SLP services are described by the service: URL and service request queries are expressed by LDAPv3 search filter. From these, SLP seems to be an attractive choice for the Internet-style service discovery.

Each service discovery protocol has chosen different query language: Java class hierarchy and assistant service attribute objects in Jini, text-based (attribute-value pair based) query in SLP, and XML query in case of UPnP. Although there are pros and cons of their design decisions, e.g., their query efficiency and expressiveness, their service advertisement and query are able to capture only static aspects of context: For example, *server-load* service attribute can indicate the load on a server machine at the moment of service announcement but not the one at the time of service query later, because it is dynamically changing. Besides, they do not provide any feature to exploit context information for service discovery.

The coverage of the three service discovery protocols is limited up to enterprise networks, because of their adoption of administratively scoped IP multicast [22]. For global networks beyond that, two service discovery protocols have been proposed. First, the wide-area extension to SLP [25], called WASRV, is trying to solve the problem by introducing BA (Brokering Agent) and AA (Advertising Agent). Wide-area services are advertised by AAs on service-specific global multicast channels. It is a BA that caches these advertisements to resolve queries from local clients. Berkeley SSDS [8] supports wide-area service discovery through a hierarchy of SDS servers and lossy aggregation of service descriptions. Service advertisements are aggregated as they propagate up along with the hierarchy.

2.2. Server Selection Mechanisms

With explosive growth of the Internet, it becomes a popular research topic to select the best among replicated services. To enable properties required for the Internet services such as scalability, load-balance, and fault-tolerance, the servers should be placed on geographically replicated locations, especially on strategic points of the network. The server selection problem is to select the best replica of those and the selection should be made transparently to clients, i.e., ideally without involving the users with it. Even though, because of the popularity of Web, most works in this research area are focused on the best Web server selection problem, their approaches show a full spectrum ranging from server side to network and to client side approaches in terms of places. With respect to layers where the selection is made, they can be classified as application-layer, DNS, and network routing layer approaches.

In DNS round-robin scheme [4], a FQDN (Fully Qualified Domain Name) is mapped to a set of IP addresses, which results in load-balancing effect among them. Since the mapping is performed in round-robin fashion, a selected IP address doesn't necessarily mean the most available server.

Several server proximity metrics have been proposed to select the nearest server. The proximity of a server and a client can be defined by *hop-count, geographical proximity, round trip time, bandwidth, and other network characteristics*. The geographical proximity proposals include SONAR service [23], DNS LOC resource record [11], and GL resource record [7]. DNS name-to-IP address resolution phase is a perfect place to implement the server selection logic, since it can be hidden from clients and does not require any change to client programs.

Smart client [34] provides an interface to a generic networked service. When a client connects to a network service, an applet to mediate an access to the service is downloaded to the client. The applet includes references to mirror sites and connects to the best server based on current load and network latency to replicated servers.

In application-layer anycast [1], a resolver keeps track of the status of a set of servers. When a client presents a URL to it, it returns the IP address of the best server based on the current load and network characteristics to the servers.

Although aforementioned mechanisms enable sophisticated server selection, they are fine-tuned to a specific type of services. As a result, they lack generality to be used by a variety of applications. To overcome this limitation, the server selection schemes are being generalized by recent RSerPool IETF drafts [26,27]. A client requests a service by sending its corresponding name to a name resolution server residing somewhere on the network. Then, the name resolver translates the name to IP address(es) and returns the list of appropriate services registered by that name. And the client connects to one of the returned services. The name-to-service translation and server status monitoring are performed by RSerPool framework for reliable server pooling. But it is more concerned with communication-oriented aspects, which positions itself as a separate entity from generic, service-oriented service discovery framework. It contrasts our approach, i.e., a generic service discovery framework that is also capable of supporting various server selection mechanisms through context attribute explained in the next section. By integrating them into a single framework, it enables a single-step service discovery and server selection.

3. CONTEXT-AWARE SERVICE DISCOVERY

3.1. Overall Architecture

The goal of our research is to provide the most appropriate service to mobile users by exploiting any meaningful contextual information. User mobility commands support for wide-area service discovery as well as local service discovery, which sets our horizontal coverage to be the Internet. With respect to the layers our proposed architecture relates to, its focus is placed on lower layers, non-application-specific context such as user location, server load, device capability, and network condition. In contrast, Web service discovery is more concerned with high-level, application-related aspects, e.g., finding a supplier offering the best price.

In the era of ubiquitous computing, computing devices are everywhere. They are interconnected through the ubiquitous computing infrastructure to help users perform their everyday tasks. Interactions between them are modeled as services. Each of those devices provides services to others so that more value-added services can be composed from primitive ones. In short, our vision is that ubiquitous computing will flourish by enabling the scalable and manageable explosion of services.

Table 1 shows three categories of services according to their coverage: *proximity service, domain service,* and *global service*. The proximity services mean services found through the closest service registry, while the domain services are ones discovered within an administrative domain boundary. The global services include any other services from the Internet. Notice that relevant context for service discovery and selection varies across the three classes. In case of proximity service, physical location or distance is likely to be the primary concern of clients, while communication or computation oriented context is usually key to effective global service discovery.

Service class	Coverage	Primary selection criteria	Example
Proximity service	Local area	Proximity	Nearby printer
Domain service	Domain	Proximity or service features	Area guide service
Global service	Global area	Server performance, user memory, or availability	Media service storage service

Table 1 - Service classification by coverage

As users search for services, their queries would end up with much more services than they can manage or need except for the proximity service case. This is because, in the future ubiquitous computing world, there would be plenty of service instances beyond their local area. We argue that this deluge of services must be dealt with by the infrastructure, since it is often the case where user devices don't have enough resource to afford to do that. The users first connect to a nearby proximity service registry to resolve their service requests. As described in detail later, an important component of the registry is *BA (Brokering Agent)* that functions as a broker to match and recommend appropriate services among a possibly large set of service instances. We note that the proximity service registry is the base-level components of our system. Our domain service registries are organized in a hierarchical fashion for beyond-the-proximity service discovery. In addition, we have introduced an entity for global service discovery, called *GSR (Global Service Registry)*, that comprises a global hierarchical tree. The GSR is an entry point to the global hierarchy through which some proximity/domain services are advertised to the globe. In other words, it is a gateway to the wide-area service discovery network on behalf of the domain it belongs to. Also, global services are discovered for local clients via it.

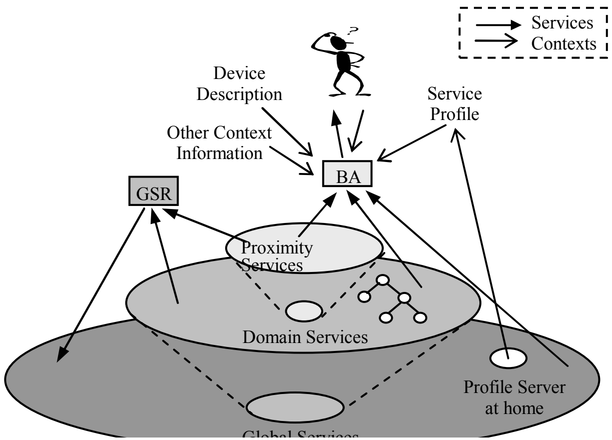


Figure 2 - Conceptual model of context-aware service discovery system

Figure 2 shows the overall view of our context-aware, integrated service discovery architecture. Proximity services are closest to a user in the figure, indicating that they, if there exist services that satisfy the user's query, are the most likely candidate services, unless explicitly specified otherwise. Domain services would be the next choice, in case the query is not satisfied by services from the proximity, and finally global services are searched for. The decision of the 3-tier approach has been made from our understanding that the primary, relevant context in the three classes is different according to the coverage. However, the three classes are hidden from the user. The user interacts with our service discovery framework via her BA. Inputs to our discovery process include various contexts such as location, network condition, server computing load, user device description, user service profile, and service query. The figure also shows that some of proximity and domain services may be promoted to global services through GSR.

Our architecture is built by adding context-awareness support to existing, general service discovery framework. The context-awareness is achieved via our context attribute evaluation by the aforementioned BA. The context attribute is discussed in the next sub-section. It provides a flexible, general way to exploit relevant context information including spatial and temporal context of all parties involved such as clients, servers, and network condition in between them. In other words, it makes use of various but relevant contextual information during the discovery process in order to reduce query result to qualified service(s).

3.2. Context Awareness Support

As discussed in the previous sub-section, primary context information relevant to service discovery is dependent on the coverage such as proximity, domain, and global areas. Besides, context handling must be transparent to the mobile users. It is our *context attribute* that provides a viable solution to this context-awareness problem.

3.2.1. Context Attribute

The *context attribute* is a special kind of attributes that are parts of service announcement messages. It is also called *dynamic attribute* in that its actual value is dynamically determined at the time of evaluation, compared to *static attribute* that has a fixed value as in existing service discovery protocols. The idea of the context attribute draws from client-side server selection mechanisms, e.g., Smart client [34], but it is generalized for service discovery in our work. It embodies service-specific selection logic hidden from clients. Also, it may prescribe conditions that should be met to further propagate throughout the service discovery network. The followings are some examples of the context attribute:

- *Distance to server*: It could describe different metrics depending on specific service implementations, e.g., geographical distance to the servers or distance in logical, organizational view.
- *Network distance*: Network distance is different from the above distance in that it is constantly changing every time it is measured. It may be measured in terms of hop count, round-trip time, bandwidth, and so on. For instance, available network bandwidth is the most critical factor for media services, whereas round-trip time for interactive applications to exchange short-messages.
- *Server load*: It is an important aspect especially for computation-intensive services. Note that the server load can be advertised as a static load attribute. But it requires too frequent updates to keep up with the load changes on the server machine, which may be problematic in wide-area discovery network.
- *Service channel*: The context attribute may look into service channels to recommend better service instances. For instance, a service *s* uses HTTP as its access protocol. If two instances of *s*, i.e., *s'* and *s",* are available, the context attribute may fetch their first Web page to see if they are cached around. Then, it recommends the one with faster response.
- *Client device capability*: Some services may put specific requirements on client device capability to use them. It is also possible for others to provide specialized support for a specific class of client devices.
- *Service advertisement constraint*: Some services may set certain range for their advertisements to be propagated. For example, a service wants to serve *foo.com* domain and to remain unknown to others to avoid being bothered by unintended clients. This special type of context attributes is called *scope attribute*, explained further in sub-section 3.2.2.

It is important to know that services provide their context attributes and clients can simply evaluate them without any knowledge of their internals. The rationale behind this is that services can be designed so they know what selection criteria matter to them the most. Also, the clients should be relieved of the burden to know the selection criteria of all types of services. The usefulness of the context attributes is highlighted especially for domain and global service discovery, since gain by the choice of the right service is magnified due to the diversity of service implementations, devices, and networks and network uncertainty. The context-attribute evaluation is usually a cooperative effort of the servers and clients, which may cause communication in between them during the evaluation process.

Benefits from exploiting context information for service discovery through the context attributes, which is a key feature of our context-aware, integrated service discovery architecture, are summarized as follows:

- It provides a general, flexible means to enable sophisticated service selection. It is independent of particular services, so service authors are given full flexibility to express selection criteria specific to their own services.
- The context attribute is an effective indicator of service quality that is able to capture any service selection criteria, including application-related aspects. It enables us to directly leverage known server selection mechanisms.
- The evaluation cost of context attributes is amortized over multiple clients. To justify the non-negligible overhead of late evaluation, the evaluation result is cached so that it can be reused for future requests for the same service, made by other clients at the local site.
- It is transparent to clients. A client only sees her local service registry. Anything else happening behind the scene is hidden from the client: the reduction of candidate service set by the context attribute evaluation and service sources, i.e., proximity, domain, or global discovery network.

3.2.2. Implementation of Context Attribute

DiscoveryCxtEvaluator and *AnnouncingCxtEvaluator* Java interfaces in Figure 3 are to be implemented by context attributes for service discovery and advertisement, respectively. All context attributes are required to implement either of them so that our BA or service registry can evaluate them by simply invoking the *evaluate()* or *isOutofScope()* method without being aware of their internals.

The first *evaluate ()* method returns a *CxtEvaluation* instance that contains *validUntil* and *suitabilityIndex*. The *validUntil* field indicates a time period during which the evaluation result is likely to remain valid. This cache feature enables the evaluation result sharing among local clients at the same site. Finally, the *suitabilityIndex* field represents the comparative suitability index of the services in question in a numeric form, for example, an index value out of 100. The *autoRefresh* parameter indicates automatic, periodical re-evaluation. If it is set to true and necessary, the context attribute is automatically re-evaluated, before its evaluation result expires. The second *evaluate()* methods takes a list of the whole candidate service instances, including itself, that implement the same type of services. As a result, it returns a list of *CxtEvaluation* in a sorted order according to their relative superiority. This evaluation process is repeated for the context attribute of each candidate, until we reach the majority of the recommendations. The idea is similar to voting to get more precise evaluation. It reduces the chance of errors due to incorrectly-implemented context attribute from a minor services. It also prevents a dishonest service from misrepresenting itself to attract more clients.

```
 public interface DiscoveryCxtEvaluator {
      CxtEvaluation evaluate(boolean autoRefresh);
       CxtEvaluation[] evaluate(boolean autoRefresh,
                                  List srvInstances);
}

public interface AnnouncingCxtEvaluator {
      boolean isOutofScope();
}

public class CxtEvaluation {
      long validUntil; // valid time set by service
      long suitabilityIndex; //indicate suitability
}
```

Figure 3 - Java interface of context attribute

The *AnnouncingCxtEvaluator* interface defines the context attribute for service advertisements as explained in the following sub-section. The context attribute can be realized in the form of procedural scripts rather than Java objects. Although it may help reduce service discovery traffic, it introduces the need of common libraries to evaluate the context script between server and client sites.

3.2.3. Context-aware Service Advertisement

The *AnnouncingCxtEvaluator* interface is used to indicate service providers' intentions as to the boundary to which their services are to be advertised. It specifies the pre-condition of potential client site context to be satisfied for announcements there. This type of context attribute is named as *scope context attribute*. If a service advertisement propagates beyond its intended area, the *isOutofScope()* method returns false. As a result, the advertisement packet is discarded at that point and does not further propagate. The purpose of the scope context attribute is a fine-grained control over the area in which the service is to be advertised. More specifically, it enables a service provider to publish its service to a certain audience, so the server resources are dedicated to its prospective clients in a productive way by avoiding being frequently disturbed by unintended customers. Otherwise, they won't find that the service is not being marketed to them, until they connect to it after the discovery and are rejected by the service application authorization logic.

3.2.4. Context-aware Service Discovery

In a domain or especially global service discovery system, it is likely for a site to be inundated with a large number of service advertisements of the same type from the globe. Although they can first be screened by a static service query matching specified by users, it will end up with still unmanageably many service instances. Therefore, it is imperative that service discovery framework be able to support a sophisticated brokering mechanism for further winnow. The need is amplified in case of resource-constrained user devices, since it is too painful for such devices to go through all candidate services to find out usable ones.

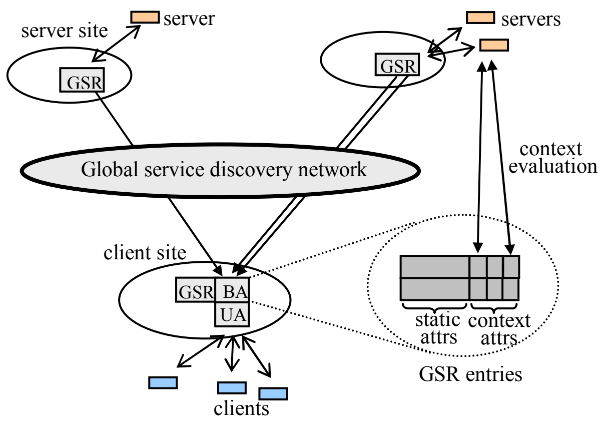


Figure 4 - Evaluation of context attributes

Figure *4* shows how the context attributes are used in our global service discovery architecture whose details are presented in section 4. Note that they can also be used with domain services, although the figure depicts only global service usage case. At two server sites, a GSR, an entity responsible for importing and exporting global services, publishes services marked as global to wide-area discovery network. The advertisement message includes string-based service description including service type, server address, and static service attributes. On the other end, they are retrieved by GSRs at client sites. Upon service requests from the clients through its UA, a BA searches the global discovery network for services satisfying static-attribute-based query in text made by the clients. This search is performed through its GSR. After then, using candidate service information from this static-attribute-based query result, the BA downloads associated objects such as service proxy and context attribute objects from the source GSRs publishing such services. Then, the candidate services are ranked and further screened through the evaluation of downloaded context attribute. But this two-step discovery by static attributes and context attributes introduces non-negligent cost. Therefore, the downloaded objects and evaluation result are cached in GSR records for the future requests from other clients at the site.

Infrastructure support for value-added service discovery is also found at [35] to propose *Sort* and *Bound* extensions to SLP. Their approach is to rank service instances according to simple static-attribute-based condition specified by clients. This selection process is solely performed by DA and servers are not involved. Smart Client [34] first suggested client-side service selection idea but it assumes the Internet services hosted on a cluster of workstations. Another interesting research is a media service discovery [33] where service discovery framework is used to cache end-to-end quality of services. Unlike the above SLP extension, our approach is able to capture dynamic aspects of context to enable much more sophisticated selection. Consequently, it achieves both the specialty of server selection mechanisms and the generality of service discovery protocols.

3.2.5. Device Description as Context Attribute

A full array of devices, including powerful workstations, PCs, PDAs, smart-phones, and pagers, usually lives in the mobile computing environments. It enriches services available to mobile users but, at the same time, disparate device capability brings a new challenge to light. For instance, a service written for desktop computers may not be able to be invoked on smart-phones. Therefore, device capability is another invaluable context information to be considered in the process of mobile service discovery.

Clients supply their device capability to a BA through their UA along with their service queries. The device capability description includes device classes, computing resources, I/O devices, and graphic toolkits. Then, the BA tries to match the client device capability as well as query itself. As a result, mobile users may see a different set of instances even with the same service query, as they switch to a different device. In order to broker services given the client device description, the BA has to figure out the-other-end requirement, i.e., device capability anticipated by service implementations. It can get such information from the description files shipped with services, e.g., JAD files of J2ME MIDlet services. Otherwise, it has to rely on Java reflection mechanism to infer the requirement.

3.2.6. User Mobility Support

A user should be able to use any kind of devices that happen to be with her at the moment: She uses desktop computers at work, her car navigation system while driving, and her smart-phone while walking. In addition, it is expected for her to be able to access the same services across different locations as well. She will expect the virtually same set of her frequently-used services, regardless of whatever devices she is using and wherever she is. But she must not be bothered to specify all detailed query terms to make a discovery of services that she is used to. Our *user service profile* enables this service mobility. It consists of a set of frequently used services, i.e., her active service set, and associated rules to re-evaluate the list. At a foreign site, the user starts interaction with her UA by specifying her home profile service from which the UA downloads her service profile. Then, it re-evaluates the service list by help of its BA to ensure that she is presented with qualified, usable services in her new environments.

4. SERVICE DISCOVERY ARCHITECTURE

Suppose that a mobile user visiting a new place needs a service on that local network. First, the user will try to discover it using small-area service discovery protocols like Bluetooth SDP, since services found nearby are likely to be superior to distant ones in most cases. If not found, she will search bigger areas and, after then, global discovery networks as her final resort. There is little well-established infrastructure support for this expanding discovery scenario by existing frameworks. That is, the user is currently left directly exposed to the choice of several service discovery technologies. She must understand several service discovery protocols, which can't usually be met by small client devices. Recognizing this problem, we propose a system architecture that provides a unified view of service discoveries with different coverage. Our service discovery subsystems for each coverage class are described below.

4.1. Proximity-based Service Discovery

A constituent element of our discovery system is the proximity-based service discovery framework that typically consists of service registries, services, and clients. They are within the reach of one another and service discovery activities are centered on service registries.

Based on general directory-based discovery architecture, our model introduces a brokering entity, called BA, added to the service registry. It can be either a component of the registry or a separate but co-located entity. The BA handles inter-registry communication and context attribute evaluation. The former is for domain and global service query routing, while the latter is to filter disqualified services given the client context. Both are explained in detail later. Since the BA takes care of all of these, the client doesn't need to know what's happening behind the scene to support seamless discovery and context-awareness.

Finally, we note that the proximity service registry becomes a node of the hierarchy of domain service registries explained in the next sub-section.

4.2. Domain Service Discovery

Although user intent may be easily captured by the implication of proximity in the proximity-based service discovery, it is often the case where non-local services are in need. As an example, assume that all services in each room are registered with its own proximity service registry. A printer in the next room might be still useful to a visitor, in case there is no printer service found in her current room. This example clearly illustrates the need of intercommunication of service registries.

The coverage limitation of proximity-based service discovery can be overcome by forming a hierarchy of service registries, which is usually based on location context [18]. More specifically, each site maintains a service registry serving that area and the registry becomes a child of the parent registry whose jurisdiction subsumes the child's. Some of services registered with their local registry are advertised beyond the registry's coverage to larger organizational domain or area. Since the upward propagation is done by the service registry tree itself, all each service needs to do is to register itself with its service registry node and to declare itself as a domain service. Note that an administrative authority is in charge of manually setting up the hierarchy of service registries within its jurisdiction. On discovery requests from clients, a local proximity service registry is first searched. If not found, the parent registry is tried. It is repeated until the search reaches the root registry for the administration domain. By this search scenario, some contextual information related to the client queries and desired services may lose their accuracy to a certain extent, as they travel up beyond their origin sites. But this incremental search scheme guarantees that discovered services will be the best among services available in the client situations, even though they may not be absolutely ideal ones.

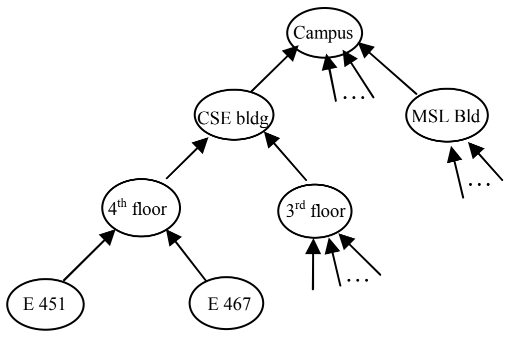


Figure 5 - An example of service registry tree

Figure 5 illustrates the hierarchy with an example of on-campus service registry tree. Suppose that each (sub-)domain has its guide service for visitors. When a visitor on the 4th floor of CSE building searches for a guide service, it is implied from her location context that she wants a floor map service for the 4th floor, unless stated otherwise. If available, it would provide the best guidance service to her. Otherwise, a CSE building guide service found in the parent node may be able to help her. The last resort would be the campus node. A service registered there may not provide any map for the 4th floor but some other useful information may be provided. This way the accuracy of the context information is gradually lost, as it gets away from the origin of the requests.

These kinds of services are called *domain services* in that they are intended to be visible only within a certain administration domain. Service advertisement and query routing for them is properly handled by the tree itself to provide the view of domain coverage.

4.2.1. Advertisement of Domain Services

Once a service provider indicates that it is supposed to be used by users in a certain domain, the service advertisement is propagated up the domain registry tree. It continues up to the root node, unless it is dropped by the evaluation of *scope context attribute* explained in sub-section 3.2.3. The service is further published to the world at the root node, if it was set to be a global service. Refer to sub-section 4.2.3 for the details of this global service promotion. Suppose that a guide service is being registered with the *4th floor* service registry in Figure 5 and it is marked as a domain or global service. Note that, by nature, a global service is considered to be also a domain service. Then, its advertisement will be forwarded to the *campus* registry via the *CSE Bld* registry. All registries on the path from the *4th floor* to the *campus* registry store it for the purpose of resolving queries from local clients or its sub-tree. There might be a concern that upper nodes are flooded with services from lower nodes. But it would not so severe, since only a portion of services is to be designated as domain or global services. Furthermore, we expect the domain size to be kept to a proper extent: A huge domain will be divided into several manageable sub-domains to be beneficial from the domain service discovery architecture.

4.2.2. Domain Query Routing

Service query routing is straightforward, since all domain or global service advertisements are propagated to upper registry nodes. A service registry forwards to its parent the queries that can't be resolved by itself. Services found at upper registry nodes are copied to the client registry which will finally return them to the user. It is to make her see the same behavior regardless of whether it is a proximity, domain, or global service. The BA of her registry will take care of this inter-registry communication.

4.2.3. Promotion to Global Services

Both domain and global service advertisements propagate towards the root of the service registry hierarchy, starting off a base proximity registry. It is further exported to global discovery network from the root, if it is marked to be a global service. For this purpose, a GSR resides at the root node. To services within the domain, it can be viewed as a proxy that publishes services to the global area on behalf of them. In other words, the root node becomes a node of global service discovery network.

4.3. Global Service Discovery

Besides Berkeley SSDS [17] and SLP WASRV [25], there is yet another noteworthy approach to the global service discovery problem: Whois++ index service [13,31]. Their directory mesh is composed of Whois++ servers as base nodes and index servers that summarize their child Whois++ servers or index servers. A layer of the mesh summarizes the next lower layer. As such aggregation method, they introduced the *Centroid*, which is a collection of every word that appears at least once in attribute values. It is propagated up the tree to provide query routing information. The Centroid at upper layers eventually converges to a large, fixed set of words, which means the global scalability.

But it is too expensive for every single word to be propagated all the way up to the top layer of the mesh, even if it occurs just once throughout the whole hierarchy. If the Centroid does not converge to a manageable size, the problem becomes severer. Since there will be plenty of services available everywhere in ubiquitous computing world, the effort to update the globe on every dynamically changing service will not paid off. Also, common, popular attributes (and their values) will be more frequently used for query terms, e.g., queries to look for a *color ps* printer rather than a printer with *a unique name*. Therefore, general attributes need to be favored over specific attributes to improve global service discovery performance without generating much traffic from the entire global network view. These facts lead us to our *localized service propagation* protocol.

4.3.1. Localized Service Propagation

Although our global services are described by a set of attribute-value pairs, just as with Whois++ servers, our LSP (*Localized Service Propagation)* approach is stingier in propagating aggregated service descriptions. The propagation of service descriptions is localized, in that it fades out, as it propagates up the hierarchy of our global service registries, and it deceases beyond a certain range. But dominant attributes are far further propagated to attract more queries, which yields better overall query performance. By dominant attributes, we mean the service attributes that occur more frequently than others at registry nodes. To describe our scheme precisely, we need the following definitions:

- *Attenuation factor,* denoted as *att,* indicates the decaying rate of the aggregation of service attributes, as it travels up the hierarchy.
- Let *n* be the number of occurrences of an attribute value at leaf nodes. This value gradually decreases due to decay process at intermediate nodes. We note that an *n* value is associated with each attribute-value pair.

The aggregation process at a registry node is to first add up all the *n* values from its children and itself attribute-by-attribute. Before passing the aggregate service description up to its parent, new *n* values, *n'*, are calculated as follows:

$$n' = \begin{cases} n*\beta \text{ where } \beta=(att\text{-c})/att & \text{if } n*\beta > 1 \\ \lceil n*\beta \rceil - 1/att & \text{if } n*\beta \leq 1 \text{ and } n>1 \\ n - 1/att & \text{if } n \leq 1 \end{cases}$$

From this formula, we see that the attenuation factor indicates the number of hops that consumes a unit value of *n*. The first line means that the *n* values of dominant attributes are decreased exponentially to prevent them from being propagated too far, while the second and third line ensures that any service information propagates up to at least *att-1* hops. According to this scheme, service information in the hierarchy attenuates, as it propagates upwards. When the *n* value for an attribute value drops below zero, it is removed at that point; hence localized service propagation. Also, note that dominant attributes are favored to propagate far further to attract queries traveling on the network. Our decaying aggregation allows us to trade off query routing cost for service advertisement traffic and the storage need of service descriptions at the upper tiers of the hierarchy.

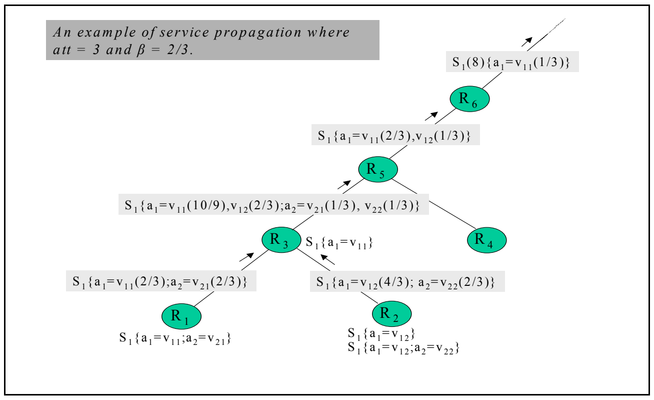

An example of service propagation where att = 3 and β = 2/3.
$S_1(8)\{a_1=v_{11}(1/3)\}$
R_6
$S_1\{a_1=v_{11}(2/3),v_{12}(1/3)\}$
R_5
$S_1\{a_1=v_{11}(10/9),v_{12}(2/3);a_2=v_{21}(1/3), v_{22}(1/3)\}$
R_3
$S_1\{a_1=v_{11}\}$
R_4
$S_1\{a_1=v_{11}(2/3);a_2=v_{21}(2/3)\}$
$S_1\{a_1=v_{12}(4/3); a_2=v_{22}(2/3)\}$
R_1
$S_1\{a_1=v_{11};a_2=v_{21}\}$
R_2
$S_1\{a_1=v_{12}\}$
$S_1\{a_1=v_{12};a_2=v_{22}\}$

Figure 6 - An example of computing the aggregate of service descriptions

Figure 6 illustrates the aggregation process of services registered in a sub-tree of the global service registry hierarchy, where *att* = 3 and β = 2/3. Each node sends its service summarization up to the parent. The annotation used in the figure denotes a service type, attribute names, pairs of attribute values and associated *n* values. For example, $S_1\{a_1=v_{11}(2/3);a_2=v_{21}(2/3)\}$ above GSR node R_1 indicates that (1) service instances of type S_1 are being registered in the sub-tree with R_1 rooted; (2) one attribute a_1 has its value v_{11} and *n* value 2/3; (3) another attribute a_2 has the *n* value 2/3 associated with its value v_{21}. Note that this service information is dropped, when it uses up the energy to advertise its presence further to upper nodes. For instance, attribute $a_2=v_{21}$ from R_1 is removed by R_5.

Our aggregation method becomes similar to the Centroid, when *att* = ∞. The difference is that our scheme remembers attribute values in its entirety, while the Centroid keeps every word occurred in attribute values.

Another important issue related to the global hierarchy is how to build it up. One solution is manual setup: The tree can be constructed via manual configuration by administrators. But it is not a good approach for large, dynamic network. As suggested by Satish Kumar *et al.* [21], the hierarchy construction through leader election algorithm seems attractive. A node periodically sends an election message with its TTL set to a certain radius. One of nodes within the range is elected as a leader and it becomes the parent of the rest of them. Then, the leader is promoted to a member of the next higher level and competes again to become a leader at that level. The TTL value of the election message is, now, doubled to reach all candidates at the current level. This process is repeated until they have a single root node, i.e., until the tree is completely constructed.

4.3.2. LSP Query Routing

Given a client query, our global service hierarchy routes it to service registries that contain matching services. The service aggregations kept by each node provide an answer to which sub-tree leads to service instances matching the query. For this, a registry has to keep a separate aggregate for each link to its children. However, the service aggregates are not always able to provide the answers, since our aggregation scheme localizes service propagations. In this case, we have to rely on *query overlay hierarchy* explained below. Our primary design goal is better query performance for queries specified in popular terms and acceptable query performance for specific queries without excessive resource use for service advertisements and updates. The rationale behind the adoption of the LSP scheme is that queries tend to be specified using popular, common attributes and their well-established values rather than very specific query terms, since clients usually do not have enough information about services in advance according to dynamic service discovery scenario. In other words, clients will likely seek anonymous services providing desired functionalities, not specific service instances. Also, we argue that the performance problem of specific queries is alleviated by our underlying domain hierarchy. Remember that user queries are first tried by our proximity and domain discovery subsystem that naturally support any type of queries. Those that can't be resolved there are referred to our global discovery hierarchy. In this sense, our service discovery subsystems complement one another to become a well-balanced system as a whole.

It is guaranteed that any service information lasts at least for *att-1* hops, which is called *propagation TTL* in this paper. Therefore, we know that there is no such a service within the TTL range from a registry without any information about the service. To improve query resolution performance, our global registry nodes at every propagation TTL distance form the query overlay hierarchy which provides faster route than the base hierarchy.

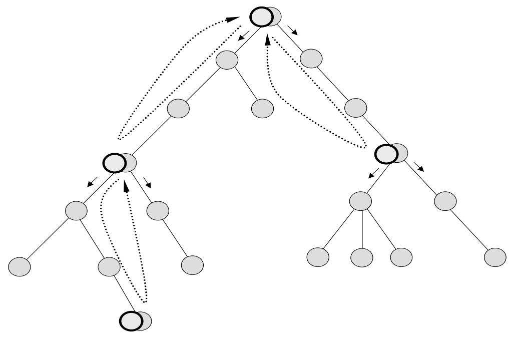


Figure 7 - An example of query overlay hierarchy where att=4

Figure 7 illustrates how the query overlay hierarchy is formed. Starting from the root, an overlay node sends a beacon downstream with the propagation TTL set to 3. Then, a child node at the TTL distance knows that it has to participate in the overlay hierarchy and responds back to the beacon source. This process is repeated, until it reaches the bottom of the base hierarchy. In the figure, thick nodes are overlay nodes.

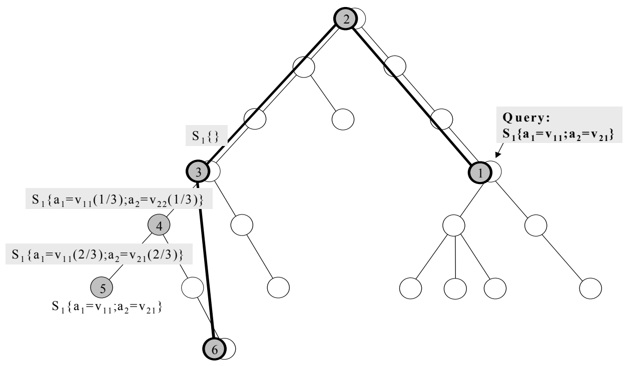

2
Query:
$S_1\{a_1=v_{11};a_2=v_{21}\}$
$S_1\{\}$
3
1
$S_1\{a_1=v_{11}(1/3);a_2=v_{22}(1/3)\}$
4
$S_1\{a_1=v_{11}(2/3);a_2=v_{21}(2/3)\}$
5
$S_1\{a_1=v_{11};a_2=v_{21}\}$
6

Figure 8 - An example of query routing where att=4

Figure 8 shows a sample query routing in our global service registry hierarchy with *att* = 4. In the figure, the query overlay hierarchy is superimposed on the base hierarchy. It shows a path to a matching service at the left bottom node, taken by a query injected into the right middle of the tree. Nodes that participated in the query resolution are shown in gray and the number on them indicates the order in which the query visited them. Starting from the query source node, the node numbered as 1 knows that the wanted service does not exist in the sub-tree by consulting its service aggregates. Then, the query is elevated to the overlay hierarchy and forwarded to the node with 3 on it through the root node. It knows that the service is in the left sub-tree from the service aggregates it keeps. From there, the query switches to the base hierarchy and eventually reaches the node numbered as 5. The figure shows that the service is discovered in 4 hops.

4.3.3. Putting It All Together

Figure 9 illustrates how the components described so far are threaded together to provide a single, integrated view of the discrete service discovery hierarchies. We note that our domain hierarchy comprises our base proximity service registries, while our global hierarchy is made up of the root nodes of each domain hierarchy. Global services are advertised into our global discovery network by the GSR-capable root of a domain hierarchy. The GSR-capable root means the root node augmented with a GSR entity in charge of exporting and importing global services on behalf of its domain. Although the GSR entity typically stands by the domain root, multiple GSR entities can be deployed at strategic points of the hierarchy where the demand for global services is high, especially in case of big domain networks. Consider the registration of a service marked as global. The first GSR entity that it encounters on its way to the domain root will advertise it to the global discovery network. Note that, by default, a global service becomes a domain service so that it must be also propagated up to the domain root. On the other hand, our query handling mechanism is aware of possible multi-GSR configuration: Suppose that a user makes a request for global services that do not exist within her domain. Then, according to our architecture, her domain hierarchy is first searched. During the query forwarding by the incremental search scheme explained in sub-section 4.2, all GSR entities on the path to the root are appended to GSR-capable registry list in the query message. When it reaches the root, it is passed on to the first GSR-capable registry of the list which is the entry point of the query to the global hierarchy.

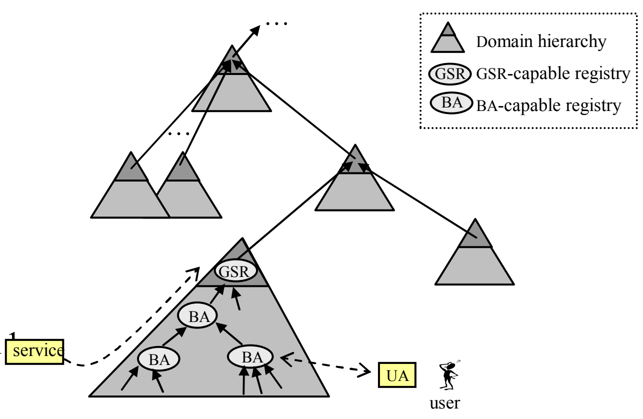


Figure 9 - Integration of domain and global service discovery framework

In our architecture, every service registry is basically BA-capable, meaning that it is capable of evaluating the context attributes to recommend appropriate services. The figure also shows that a user interacts with a service registry closest to her, through her UA, and services may propagate up the domain hierarchy or even up to the global hierarchy.

5. SIMULATION OF LOCALIZED SERVICE PROPAGATION

We did simulation to compare the performance of our LSP and the Centroid, since both are using string-based attribute-value pairs for their service descriptions. Performance comparison between the Centroid and Berkeley SSDS was reported in [17]. They achieved a good scalability by employing the Bloom filter as their aggregation method.

5.1. Simulation Objective

The objective of our simulation is to validate the design decision of our LSP: Achieving acceptable query performance comparable with other global service discovery protocols without using excessive resource for service advertisements. Note that the LSP performs better than others on the networks with the uneven, clustered population of similar services. Also, it outperforms the Centroid in a sparse node topology. However, any favor is not given to our LSP for the simulation objective: Our LSP trades query performance for service advertisement cost, and we want to analyze the inefficiency of query routing due to its localized service propagation. We choose a dense tree and even service distribution. In fact, this simulation setting is close to the worst condition for our LSP.

The global hierarchy for the simulation is a complete binary tree comprised of 255 nodes. As our workload, we use the freedb CD database (http://www.freedb.org) as in [17]. The record for a CD consists of several, text-based attributes and their values. Among them, we classify *YEAR* and *GENRE* as general attributes, popular for search queries, and rest of attributes as specific ones. Examples of the specific attributes include *DISCID*, *DISC TITLE, ARTIST, and SONG TITLES*, which have very unique values. We compare our LSP with the Centroid in terms of query cost and storage used for service advertisements. Note that the storage need is proportional to the traffic of information exchanged between service registries for service advertisements and updates.

5.2. Simulation Results

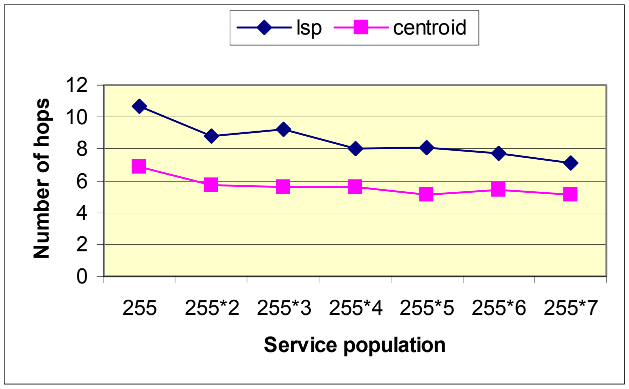


Figure 10 - Query cost according to service population

Figure 10 compares the query performance of the LSP and the Centroid for different service population. Again, our simulation network topology is a complete binary tree with 255 nodes. Total 255 queries are fed to each run of our simulation, whose general to special query ratio is 1:1. The services and queries are assigned to specific nodes at random. Also, the LSP simulation is done with *att*=5 and β=3/5. Under these simulation settings, the result shows that the Centroid always outperforms the LSP as expected. This is so because the Centroid that keeps all service information provides nearly optimal query performance (There were a very small number of *false positives* witnessed in our simulation).

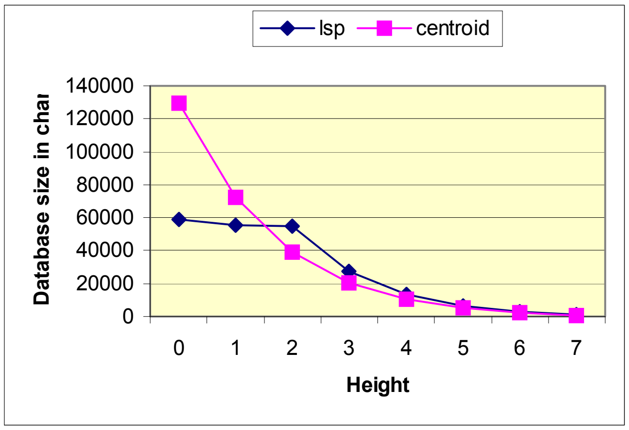
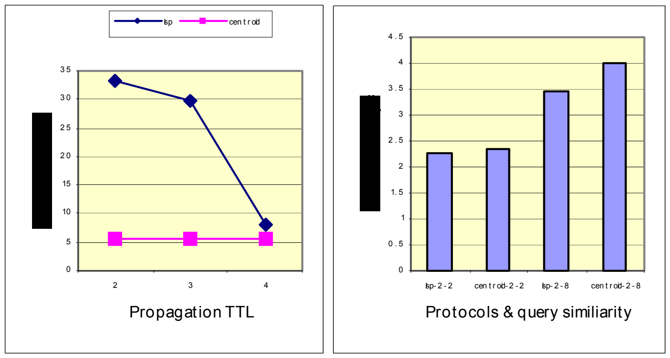


Figure 11 - Database size at each layer

Figure 11 is the simulation result, when 255*5 services are randomly populated on the complete binary tree. Other configurations and conditions are the same as in the previous simulation. The graph shows the average service aggregate size that a node keeps at each layer. In case of the Centroid, the database size grows exponentially, even if duplicated words are dropped as per the aggregation method. The gain from this aggregation was negligible with the workload size of 255*5 services. The service database size of the LSP converges in certain layers, depending on the *att* value, because of its localized service propagation scheme. The LSP is worse than the Centroid below height 2, which accounts for the overhead of the *n* values associated with each attribute in the LSP.



(*a) Propagation TTL* *(b) A sparse tree*

Figure 12 - LSP query performance

Figure 12 compares the query performance of the LSP and the Centroid under different conditions. Graph (a) shows LSP query performance for the propagation TTL values of 2, 3, and 4. The Centroid performance is additionally shown as the base line for comparison, although it is independent of the TTL values. Other conditions are the same as the simulation in Figure 11. As the TTL value becomes smaller, the LSP performance becomes drastically worse. For the TTL value of 2, the performance is similar to an exhaustive search of the half size of the simulation tree. This is because the query overlay hierarchy grows correspondingly. Therefore, the TTL value, i.e., *att -1,* should be carefully determined considering the global service discovery hierarchy size as well as the trade-off between storage and query performance.

Graph (b) compares the query performance of both schemes on the network shown in Figure 8, with general to specific query ratio of 1:1 and 1:4, respectively. In this special topology, the LSP query performance beats the Centroid's performance.

Finally, we ran another set of simulations to evaluate query performance with various ratios of general and specific queries. The simulation result showed that the number of hops increases gradually in both the LSP and Centroid cases, as the portion of specific queries grows.

6. CONCLUSION

The main ideas presented in this paper are the *context attribute* and *multi-tier service discovery architecture*. The context attribute is an effective means to capture relevant, dynamic context related to client, service, and network condition in between them, in connection with dynamic service discovery. Since it provides a framework for any service-specific selection logic, the context attribute achieves both the specialty of server selection mechanisms and the generality of service discovery protocols. We designed a service discovery architecture for mobile clients, consisting of proximity, domain, and global service discovery sub-systems. This 3-tier architecture is based on different selection criteria and requirements for each sub-system. The context attribute and the 3-tier architecture concepts are perfectly incorporated into our service discovery framework to well address the need of dynamic service discovery in the future world of ubiquitous computing.

We have designed Localized Service Propagation (LSP) protocol as part of the 3-tier architecture. We compared our LSP protocol for wide-area service discovery with the IETF Centroid approach through simulation. The simulation result verified our concern: The LSP protocol query performance is within acceptable limits when compared with the Centroid. We also verified that LSP reduces service advertisement cost.

We plan to develop a prototype of our 3-tier service discovery architecture. The context attribute just defines a general agreement and its real implementation is left to service authors. Also, its evaluation process may require cooperation across network, between clients and services. Java object and RMI perfectly fit into these needs. Thus Jini will be chosen as the base system for our proposed service discovery architecture prototype.

7. REFERENCES

[1] Samrat Bhattacharjee, Mostafa H. Ammar, Ellen W. Zegura, Viren Shah, and Zongming Fei, "Application-layer anycasting," In Proceedings of the IEEE INFOCOM '97, April 1997.

[2] James Beck, Alain Gefflaut, and Nayeem Islam, "MOCA: A Service Framework for Mobile Computing Devices," Proceedings of the ACM international workshop on Data engineering for

wireless and mobile access, August 20, 1999, Seattle, WA USA.

[3] Bluetooth Consortium, "Specification of the Bluetooth System Core Version 1.0 B: Part E, Service Discovery Protocol (SDP)," Nov 29, 1999. http://www.bluetooth.com/developer/specification/specification.asp.

[4] Thomas P. Brisco, "DNS Support for Load Balancing," IETF, RFC 1794, April 1995.

[5] Cisco, "Distributed Director," http://www.cisco.com/warp/public/cc/pd/cxsr/dd/index.shtml.

[6] Guanling Chen and David Kotz, "A Survey of Context-aware Mobile Computing Research," Dartmouth Computer Science Technical Report TR2000-381.

[7] Al Costanzo, "Definition of the DNS GL Resource Record used to encode Geographic Locations," IETF Internet Draft (draft-costanzo-dns-gl-05.txt), August 2001.

[8] Steven E. Czerwinski, Ben Y. Zhao, Todd D. Hodes, Anthony D. Joseph, and Randy H. Katz, "An Architecture for a Secure Service Discovery Service," Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking (MobiCom '99), August, 1999, pages 24 – 35.

[9] Anind K. Dey, Gregory D. Abowd, and Daniel Salber, "A Context-Based Infrastructure for Smart Environment," In Proceeding of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99)

[10] Anind K. Dey, Masayasu Futakawa, Daniel Salber, and Gregory D. Abowd. "*The Conference Assistant: Combining ContextAwareness with Wearable Computing*". In Proceedings of the 3rd International Symposium on Wearable Computers (ISWC '99), San Francisco, CA, October 21 - 29, 1999.

[11] Christopher Davis, Paul Vixie, Tim Goodwin, and Ian Dickinson, "A Means for Expressing Location Information in the Domain Name System," IETF RFC 1876, January 1996.

[12] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henrik Frystyk Nielsen, and Tim Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," IETF RFC 2068, January 1997.

[13] Patrik Faltstrom, Rickard Schoultz, and Chris Weider, "How to Interact with a Whois++ Mesh," IETF RFC 1914, February 1996.

[14] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service Location Protocol, Version 2," IETF RFC 2608, June 1999. http://www.ietf.org/rfc/rfc2608.txt

[15] Erik Guttman, Charles E. Perkins, and James Kempf, "Service Templates and Service: Schemes," IETF RFC 2609, June 1999. http://www.ietf.org/rfc/rfc2609.txt.

[16] Arnt Gulbrandsen, Paul Vixie, and Levon Esibov, "A DNS RR for specifying the location of services (DNS SRV)," IETF RFC 2782, February 2000.

[17] Todd D. Hodes, Steven E. Czerwinski, Ben Y. Zhao, Anthony D. Joseph, and Randy H. Katz, "An Architecture for Secure Wide-Area Service Discovery," ACM Wireless Networks Journal, special issue, Volume 8, Issue 2/3, pp. 213-230, March/May 2002.

[18] Rui Jose and Nigel Davies, "Scalable and Flexible Location-Based Service for Ubiquitous Information Access," In Proceedings of First International Symposium on Handheld and Ubiquitous Computing, HUC'99, September 1999.

[19] "Jini Technology Architectural Overview," January 1999, http://www.sun.com/jini/whitepapers/architecture.html .

[20] "Jini Specifications v1.1," October 2000, www.sun.com/jini/specs.

[21] Satish Kumar, Cengiz Alaettinoglu, and Deborah Estrin, "SCalable Object-tracking through Unattended Techniques (SCOUT)," In Proceedings of the 8th International Conference on Network Protocols, November 2000.

[22] David Meyer, "Administratively Scoped IP Multicast," IETF, RFC 2365, July 1998.

[23] Keith Moore, "SONAR -- a network proximity service, " IETF Internet Draft (draft-moore-sonar-03.txt), August 1998.

[24] Craig Partridge, Trevor Mendez, and Walter Milliken, "Host Anycasting Service,' IETF RFC 1546, November 1993.

[25] Jonathan Rosenberg, Henning Schulzrinne, and Bernd Suter, "Wide Area Network Service Location," IETF Internet Draft, November 1997.

[26] Michael Tuexen, Qiaobing Xie, Randall Stewart, Melinda Shore, Lyndon Ong, John Loughney, and Maureen Stillman, "Requirements for Reliable Server Pooling," IETF Internet Draft, May 2001

[27] Michael Tuexen, Qiaobing Xie, Randall Stewart, Melinda Shore, Lyndon Ong, John Loughney, and Maureen Stillman, "Architecture for Reliable Server Pooling," IETF Internet Draft, April 2001.

[28] Universal Plug and Play Device Architecture, http://www.upnp.org/download/UPnPDA10_20000613.htm, June 2000.

[29] Mark Weiser, "The Computer for the 21st century," Scientific American, 265(3): 94 – 104, September 1991

[30] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons, "The active badge location

system," ACM Transactions on Information Systems, 10(1):91--102, January 1992.

[31] Chris Weider, Jim Fullton, and Simon Spero, "Architecture of the Whois++ Index Service," IETF RFC 1913, February 1996.

[32] Roy Want, Bill N. Schilit, Norman I. Adams, Rich Gold, Karin Petersen, David Goldberg, John R. Ellis, and Mark Weiser. *An overview of the PARCTAB ubiquitous computing experiment*. IEEE Personal Communications Magazine, 2(6):28--43, December 1995.

[33] Dongyan Xu, Klara Nahrstedt, and Duangdao Wichadakul, "QoS-Aware Discovery of Wide-Area Distributed Services," In Proceedings of IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid 2001), May 2001.

[34] Chad Yoshikawa, Brent Chun, Paul Eastham, Amin Vahdat, Thomas Anderson, and David Culler, "Using Smart Clients to Build Scalable Services," In Proceedings of the 1997 USENIX Annual Technical Conference, January 1997.

[35] Weibin Zhao, Henning Schulzrinne, Chatschik Bisdikian, and William Jerome, "Customization for SLP Service Request and Reply," IETF Internet Draft, July 2001.

[36] Abdelsalam Helal, Choonhwa Lee and David Nordstedt, "Impromptu Service Discovery and Delivery for Pervasive Computing Environments", Confidential Technical Report, Pending clearance from Motorola Corp.