Resource Discovery in Wide Area Networks

by

Bruno Godin

B.Sc.(H), Queen’s University, 1999

AN ESSAY SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science
in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this essay as conforming
to the required standard

______________________________

The University of British Columbia

May 2001

© Bruno Godin, 2001
Abstract

The internet is expanding to include more than just files. New systems allow any number of devices and services to be connected to the internet. This increase means locating resources will become a even bigger challenge than it is currently. This essay surveys recent techniques in resource discovery in order to collect those that can respond to this growth. Of these, four standout: storing information in distributed hierarchical structures, searching hierarchically, naming resources with XML, and gathering data by advertise/subscribe schemes.
Contents

Abstract ii

Contents iii

1 Introduction 1

2 Naming 3
  2.1 URL ......................................................... 3
  2.2 URN ......................................................... 4
  2.3 XML ......................................................... 4

3 Gathering/Propagating Information 6
  3.1 Gathering ................................................. 6
    3.1.1 Full Discovery ....................................... 6
    3.1.2 Advertise/Subscribe ................................. 7
  3.2 Propagation .............................................. 8
    3.2.1 Gossip ............................................... 8
    3.2.2 Flooding ............................................. 9
    3.2.3 Name Dropper ....................................... 9
<table>
<thead>
<tr>
<th>4</th>
<th>Storing Information</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Central Collection</td>
<td>11</td>
</tr>
<tr>
<td>4.2</td>
<td>No Collection</td>
<td>12</td>
</tr>
<tr>
<td>4.3</td>
<td>Hierarchichal</td>
<td>13</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Compression</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Searching</td>
<td>17</td>
</tr>
<tr>
<td>5.1</td>
<td>Unicast, Broadcast and Multicast</td>
<td>17</td>
</tr>
<tr>
<td>5.2</td>
<td>Hierarchical Searching</td>
<td>18</td>
</tr>
<tr>
<td>5.3</td>
<td>Optimisations</td>
<td>19</td>
</tr>
<tr>
<td>6</td>
<td>Security</td>
<td>20</td>
</tr>
<tr>
<td>6.1</td>
<td>Sevice Location Protocol</td>
<td>20</td>
</tr>
<tr>
<td>6.2</td>
<td>(Secure) Service Discovery Service</td>
<td>21</td>
</tr>
<tr>
<td>6.3</td>
<td>Separate Discovery and Access</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>Conclusion</td>
<td>23</td>
</tr>
</tbody>
</table>

Bibliography 25
Chapter 1

Introduction

This essay provides an overview of the current approaches to resource discovery. Resource Discovery is an important issue as the growth and expansion of the internet continues. As the network expands to provide high bandwidth access to all users, the ability and the demand for shared services and network enabled devices will also increase.

The popularity of file sharing systems such as Napster, Gnutella, Freenet and more recently JXTA demonstrate that there is a wide user base for resource sharing applications. Part of the success of these systems is the ability to search for files quickly and efficiently. There are no unrelated web pages that have to be viewed and the user knows within minutes whether the resource is available or not.

The development of mobile devices and mobile LANs will continue and there is a need to provide adequate mechanism for locating and using these devices. Projects such as Bluetooth [11], Universal Plug and Play [5], and Salutation [3, 4] show that the industry believes that ubiquitous and pervasive computing is the next step in the development of networks. It is obvious that all these devices and services will require new methods for naming and locating resources.
There are many processes involved in finding and using resources in a distributed wide area environment. The location and description of resources must be stored and made available through a well known mechanism, there must be algorithms for collecting and propagating this information, and there must be methods for searching this information. In addition, information must be maintained: updated, verified and protected.

A location service should attempt to achieve certain goals. It needs to be efficient. If the system is global, any waste of bandwidth or resources becomes a serious liability. It should scale well. The growth of the internet is accelerating; the addition of network-aware appliances will only contribute to this trend. The service should be fast. Successful services are those that provide timely responses. Internet users are notoriously impatient and there is no reason to believe this behaviour will be different when accessing appliances instead of web pages. Results from queries should be relevant and in the case of services they should be correct. For example, if you requested a color printer, you expect colour output. In addition to these characteristics, there are quality of service considerations. Can I specify locality of results? Can I get statistics on recent performance/load? What about reliability and trust? A system need not require these QoS criteria of all resources but it is reasonable that it be possible to report such information.

This essay is an attempt to extract the more promising techniques and solutions to this general problem. It is split into chapters dealing with naming, gathering and propagating data, storing data, searching data, and finally security issues. The terms resource and service are used interchangeably within this document to mean any network accessed resource, service or device.
Chapter 2

Naming

Naming is integral in locating services. It can greatly influence the way a location service functions. It determines the richness of service descriptions. It influences the way the space of services can be searched, stored and compressed, which can in turn influence the quality of results and the performance of the system. For example, a richer service descriptions may return more accurate and meaningful results at the cost of extra storage and greater processing time. Additionally, the naming scheme is also a factor in terms of compatibility and ease of deployment. A scheme that requires the renaming of all resources or a fundamental change in the way resources are commonly identified requires more effort to adopt. The remainder of this section discusses these issues in relation to three naming schemes.

2.1 URL

The Service Location Protocol (SLP) [23] has as a main goal to require as little modification of the current internet as possible. Therefore, its naming scheme is an extension of the URL specification used in the World Wide Web which in turn
is based on DNS [15, 16]. DNS uses a hierarchical naming scheme meant to reflect the hierarchy of the internet protocol (IP) [19] name space. DNS relies on translation of word based names (ex. techno.cs.ubc.ca) into the numerical namespace of IP which is used to identify machines on the network. URL extends the name with the location of the resource on the machine (techno.cs.ubc.ca/file.txt) and the nature of the connection (ftp://, http://, file:/// ) that is to be made to that machine. SLP’s extension is to add a service identifier in addition to the normal connection type, thus allowing devices and other resources to be located in addition to files. The service identifier is placed similarly to the connection identifier but may be stacked. For example, a printer might be identified by the following URL: ‘/service:printer:lpri:hp306.cs.ubc.ca/draft’.

### 2.2 URN

Hamilton [12] proposes greater extensions to the URL scheme called URN. The URN is a higher level of naming which would be longer lived. There are two problems with URLs: they suffer from dangling links and they represent only instances of resources. URNs would be resolved to a list of URLs instead. The URN itself consists of a scheme ID and an element ID. It follows that commonly used naming schemes could be used to identify resources. Very little additional support is needed within the network in order to use URNs.

### 2.3 XML

Many systems (Jini, JXTA, SDS and others) are adopting XML [2] for naming and description purposes. XML provides many benefits to naming. First, it is a
well known and widely adopted standard that provides cross platform compatibility. Second, it allows descriptions of arbitrary complexity. This means it can be used to describe any service to any degree of desired detail. Descriptions can include any number of service specific tags. Third, it can be hierarchical or not, i.e., the naming scheme does not need to assume any specific organisation of location data. Resolving non-hierarchical names could require more effort, but XML gives implementors the choice. Finally, XML is self-describing. XML schemas [21] provide a description of the contents of the XML document. Since a reference to the schema resides in the XML document, anybody can process the document even if they have never encountered a similarly formatted document. This flexibility makes it an attractive choice in the heterogeneous world of the internet.
Chapter 3

Gathering/Propagating Information

A location service is only as good as the information it provides. An incorrect answer is still incorrect regardless of how fast and efficiently it is provided. Good information can, however, mean different things. One can measure it by completeness, timeliness, relevance, or quality of service. Location services attempt to achieve these goals to varying degrees. This chapter presents a few of the approaches used in this pursuit.

3.1 Gathering

3.1.1 Full Discovery

Full discovery implies exploring the whole internet and recording all service information found. This is not a small task, many web search engines are based on this principle and only capture a small amount of the available resources. For example, Google [14] only manages to capture about 25 percent of the available static pages and only 10 percent of the total pages available [18].
Harvest [1] attempts to solve the some of the inefficiencies suffered by Full
discovery systems, namely the poor timeliness and the lack of generality. To get more
up to date information, Harvest distributes the task of gathering data. It works as
follows: Gatherers in Harvest are responsible for one or more providers. In the
case of one machine, the gatherer is run on the provider’s machine itself, otherwise
the gatherer contacts the machines over the network. The Gatherer indexes service
information locally so that it may transmit only relevant information to each broker.
Brokers collect information from gatherers and provide an indexed query interface
to the data. Brokers may also collect information from other brokers allowing them
to be organised hierarchically. To reduce communication a gatherer (or broker)
only transmits new information it has obtained since the last request by the same
broker. In addition, brokers are intended to index data by topic, thus avoiding the
inefficiency of global searches.

3.1.2 Advertise/Subscribe

Many systems sacrifice full discovery in order to concentrate on the task of stor-
ing and delivering information. By using an advertise/subscribe scheme they put
the responsibility of collecting information in the hands of the service providers
themselves. Timeliness of data is assured by the desire of providers to advertise
up-to-date information. The system has to ensure that a provider does not flood
it with advertisements to the detriment of others. Another advantage of such a
system, is the potential reduction in communication. Because only those hosts that
wish to be on the service will contact it, other hosts do not need to be contacted
at all. The Service Location Protocol (SLP), The Secure Service Discovery Ser-
vice (SDS), JINI, Predator, and DNS all use some form of advertisement system to
gather information. [24, 25, 6, 23, 15]

3.2 Propagation

Unless data is stored centrally, gathered service announcements and updates need to be propagated to participants in the system. This section describes a few algorithms used for this purpose.

3.2.1 Gossip

Gossiping is a well known technique of data propagation in flat networks. Generally it means that a machine exchanges its information with a neighbour until everyone in the network is contacted. The Captain Cook [22] system uses this technique and is described the remainder of this section. It uses a simple gossip algorithm that is slightly modified to deal with a hierarchichal structure (see Section 2).

Gossip in Captain Cook functions as follows: At each round each node randomly picks an entry to propagate. The entry is timestamped with the time of its last update. The node sends the entry to a randomly picked node within its domain. The receiving node then compares the timestamp of the corresponding entry in its own table. It then either stores the recieved entry or replies to the sender with its more up to date information. If the sending node picks itself as receiver it chooses a random contact from its parent domain. This may go on until the node picks contacts in the root domain. At this point, if the node picks itself again then no message is sent. Captain Cook nodes can gossip to nodes in its ancestor domains because they hold copies of the service tables of all their ancestors. This scheme results in rapid updates within a domain and slower propagation to domains further away.
How does Gossip perform? In the flat case (i.e. within a single domain),
the number of rounds of communication is \( O(\log n) \) where \( n \) is the number of
participants. For a hierarchically based system the slowdown is greater but the
number of messages is greatly reduced. If \( m \) is the greatest number of machines
that can be supported in a flat gossip network then \( m^l \) machines can be supported
in a \( l \) level hierarchical network.

\subsection{Flooding}

In Flooding [17], machines have a fixed set of neighbours and only communicate
with these neighbours. At each round, each machine \( u \) sends all new information
it has received since the last round to all of its neighbours \( v \). All the machines \( v \)
then merge the new information to their own and the next round begins. It takes \( d \)
rounds for all information to propagate, where \( d \) is the diameter of the network. The
total number of messages is \( d \cdot m \), where \( m \) is the total number of neighbours (for
example, if 20 machines have an average of 3 neighbors each that’s 60 neighbours).
It is easy to see that this method incurs a lot of communication cost. Even so this
is the method used by internet routers.

\subsection{Name Dropper}

The Name Dropper algorithm [13] tries to be more efficient than flooding by having
each machine communicate with only one other machine. It is almost identical to
the Random Pointer Jump Algorithm but performs much better. At each round
machine \( u \) sends all its information to a random machine \( v \) that it knows about (it
is not limited to just those in the initial set of neighbours). Machine \( v \) then merges
the information. With high probability, it takes \( O(n \log^2 n) \) for all the information
to propagate through the network.

As a testament to its efficiency, Name Dropper is licensed by Akamai Technologies for their wide-scale distributed caches.
Chapter 4

Storing Information

Storing location data can be accomplished in many fashions, but the goals remain the same. The data should be indexed so as to facilitate queries and improve the quality of results. The data should be made (highly) available through replication or distribution. The data should be manageable. This chapter presents three methods of storage and how they achieve these goals.

4.1 Central Collection

Centralised stores suffer from several drawbacks. In a dynamic environment they cannot react quickly to changes. Location data must all be propagated back to a single point which means that unless there is a continuous broadcast across the network data becomes stale. This is often the case of Web Search Engines which rely on continuous crawling of the internet to discover new sites. In addition, central stores must rely on replication to provide reasonable performance and some amount of fault tolerance [8]. Replication adds more overhead to updates. A central store may also simply become to big to efficiently process requests or to replicate. For
example, Google [14] one of the more popular web search engines has a server farm of 8000 machines to deal with queries and store over a petabyte (1 million gigabytes) of data. Another option if increasing server (or server cluster) size is not possible, is to reduce the size of the data. Entries can be made to contain minimal information about a resource. Unfortunately, this will reduce the accuracy of results. Neither of these options is very appealing. The lack of adequate scalability and flexibility effectively excludes this as an option for internet wide service location.

4.2 No Collection

At the other end of the spectrum there is the case where no data collection is done at all. Client queries must propagate to the actual matching service provider before an answer is received. This is how Gnutella [9] works and it is one of the reasons why Gnutella suffers from scaling problems. In any such system, it is possible that a query may have to reach every node in the system before a result is returned. This generates large amounts of traffic as, on average, half the nodes must be reached. If queries are limited in the number of nodes they may reach, traffic may be reduced but so is the scope of the search. Latency can be either very small for a shallow multicast tree or extremely long for a deep tree; the former case being one of the main attractions of this scheme. Another advantage is that routing is simple and no processing is required. Even so the traffic demands make this implausible for the scale required for internet wide service location.
4.3 Hierarchical

Given that data is hard to manage centrally and does not scale well if data is not collected at all, many systems achieve a compromise by distributing partial information in hierarchical structures. These hierarchies can be organised by different criteria such as geographic locality, network locality, or service types. As many systems use this technique several examples are presented.

DNS [15, 16] is easily the most well known location service. Each network domain is associated with a primary server which is responsible for that domain. The domain can be further divided into subdomains. This hierarchy is not deep but allows DNS to store location data for all hosts on the internet. Location entries in DNS tables is small, this allows root servers to keep a table for all domains and respond to a query with reasonable delay (load notwithstanding). Unfortunately this hierarchy is also a bottleneck, as all queries unanswered by the local domain go to the root servers. Nonetheless DNS has been very successful and robust. Just a shallow hierarchy has enabled it to last and scale beyond the capacity it was intended to support. This success may have inspired the many other hierarchical systems described in this section.

SDS [6] is also hierarchical but includes much more functionality (some of which is discussed in other sections) than DNS. In the local area case, each server is responsible for a particular domain. Any request is made to that server. If the load on the server becomes too great, the server may create a child server and partition the domain among its children. In the Wide Area case, SDS has a similar solution to (other systems). Though not fully implemented, the authors describe a multi-level hierarchy. As mentioned previously, hierarchies can be based on many criteria. SDS addresses this by allowing multiple hierarchies within the system. SDS
servers can choose to participate one or more hierarchies but all must participate in
a primary hierarchy to insure connectedness of the servers. As with other systems,
scalibility comes from ‘lossy aggregation of service descriptions’. Servers higher up
in the hierarchy have more general information and forward requests down towards
possible matches. This is discussed further in the following subsection.

The Predator System [25] is organized in a hierarchical tree structure based
on geographic/physical locality. All clients and service providers belong to this base
tree. As in SDS, Predator allows different logical hierarchies to be defined based
on different types of services that are provided or requested. The root is the same
for all hierarchies and different services appear as logical subtrees. These different
hierarchies are built dynamically as mobile objects join or leave the system.

Globe [10] like Predator is based on global objects and does not compress
data. The hierarchy is tree of ‘directory nodes’ which represent geographical, to-
ographic or administrative regions, with the root representing the whole world.
Each service is represented as an object and a set of contacts which are machines
that provide the service. A new contact is added to the nearest directory node and
propagated up the hierarchy to the root. Since no compression is used, the root
actually holds all contact information for all objects. In order to scale the nodes
in the hierarchy can be partitioned into subnodes. The partitioning is based on the
leftmost bits of object handles. This does not address the issue of objects that have
very large numbers of contacts. Also, the actual storage needed increases by a factor
of the depth of the tree.

The Captain Cook System [22] is made up of one or more hierarchical trees.
Each tree is made up of tables. Leaf nodes contain information related to each
service in its particular domain. Internal nodes contain information about each of
its child domains). The information that is stored can be related to type of service, load, or other QoS concerns but must contain at least an ID and a contact list. Each table has a condensation function that is used to compress the table to a single row which is used as an entry in the parent’s table. This function is described in the following subsection. In addition to the table representing its own domain each node also stores a copy of the table of all its ancestors (toward the root). Storage demands increase as nodes get further from the root (as the tree expands). This extra space is traded for the ability to shortcut queries so they don’t have to make their way up the tree.

Harvest [1] uses a structure similar to DNS. There are two types of nodes in the system: gatherers and brokers. Gatherers are basically interfaces to various services on the internet. They are either run at the provider itself, even though the provider itself is not part of the system, or as a separate entity on the network. In the first case the gatherer collects and indexes service information for that provider only. In the second case the gatherer may be responsible for more than one provider. Brokers actually store the searchable indexes of service information. Individual Brokers or small groups are meant to index topic specific information. They can be layered to provide a hierarchical structure with individual brokers filtering and refining information as required. A central registry server keeps track of all brokers and gatherers, and acts as the contact point for clients. Unlike most of the other systems, there is no primary structure to link all these nodes together except for the registry. Effectively this means all the topical hierarchies are children of the registry. Because the registry is a single point of failure it is highly replicated.

All of these systems have to deal with enormous amounts of information, a popular solution is to compress data so top level servers remain manageable. The
following subsection discusses this in more detail.

4.3.1 Compression

In a hierarchical system, servers higher up in the hierarchy must be able to direct queries about anything that may reside below it. Obviously there is a need to represent more data as nodes get closer to the root. This can be tackled in two ways: increase the capacity of the server or represent the data more compactly. Since queries are typically forwarded down the hierarchy it doesn’t make sense to store exhaustive information higher up. This section presents some of the methods used to compress the data as it is propagated up through a hierarchical structure.

SDS uses a method based on hashing and summarisation using Bloom filtering. The method consists of hashing description tags to fixed length bitstrings. Because descriptions may contain multiple tags (depending on the level of detail included by the provider, see section 2.2), all the combinations of these tags would return a positive match. To reduce the number of hashes that are stored, SDS allows setting the cross products of tags that are calculated for any given description. This trades off accuracy for slightly reduced storage and much faster processing without sacrificing correctness. More false positives are reported but no true positives are missed.

To reduce storage further, Bloom filters [20] used on the hash tables. Bloom filters compress the hash tables into fixed size tables, again this increases false positives but maintains correctness. The resulting table is a bit vector. In the hierarchy, server’s bit vector is the ORing of its childrens bit vectors with its own. A server’s storage needs are dependent on the number of children it has and not the number of services that are advertised in its branch of the hierarchy.
Chapter 5

Searching

This chapter presents approaches used for querying data once it has been collected and stored. For obvious reasons the organisation of the data has a great influence on the method of searching used. Of course certain systems are structured based on client queries themselves. In this case the discovery and storage are one and the same.

5.1 Unicast, Broadcast and Multicast

Unicast is the obvious choice if all the location data resides on a central server, then searching involves running a simple search algorithm on the database. If the service is replicated, this may additionally involve finding the replicas. As mentioned in chapter 3, central databases suffer from a few problems. First, as all queries must go to the central server (or server farm) load can be very high. Second, each search must be completely processed at this same site.

In a hierarchical system load is distributed more evenly. Ideally unicast queries would be spread evenly across the hierarchy. One drawback of unicast is
that failures cause backtracking. If a query follows a branch that ends up not 
containing the correct information, the query must be sent back up to a node that 
has another candidate branch. This leads to an increased latency on individual 
queries. The main advantage of unicast is that the number of messages is kept to a 
minimum if there is little backtracking.

Broadcast guarantees that a query will be answered with the most recently 
stored information in the system. Depending on its implementation, this can also be 
the method of query with the lowest latency. Unfortunately, in a wide area network, 
flat broadcast is simply not feasible because of the communication costs.

Multicast offers a compromise. If queries are branched a lot, then latency is 
small while communication cost is higher and vice versa.

In reality, most systems are more intelligent than this and use a combination of these. This is demonstrated by hierarchical searching.

5.2 Hierarchical Searching

Hierarchical systems mostly follow the same basic search strategy. Queries are tested 
against local information and then propagated to the next level in the hierarchy. The 
parent can then send the query down to another child or up again and so on. If 
the query reaches the root and it doesn’t believe any of its branches can respond 
positively to the query, the root returns a negative response. Similarly children 
that receive queries from parents will respond with a negative if they believe none 
of their children can answer the query. Overall, queries are unicast up the hierarchy 
and then intelligently multicast down the hierarchy towards promising candidates.

Search performance is highly dependent the structure of the hierarchy. If the 
systems structure is based mostly on criteria not related to your query performance
may suffer. Performance can also be influenced by optimisations such as caching and shortcbounding which are discussed in the following section.

5.3 Optimisations

This section presents two optimisations that can be applied to the basic hierarchical location service: including Quality of Service (QoS) information with advertisements and caching advertisements at participating service nodes.

Adding QoS information enables clients to choose a service provider that more closely matches their needs. Caching improves response time, reduces network traffic, and reduces the general overhead of the system. The cost of these optimisations are more processing and storage. QoS needs to be calculated and stored. Caching requires cache lookups and the storage of the cache tables.

Xu et al [7] propose an implementation of these optimisations on top of SDS. For QoS, clientside routines are added to monitor QoS during service connections. Though QoS is measured differently depending on the nature of the service, QoS is converted to a universal value for advertisement. This allows the discovery server to remain ignorant of the nature of the service. For caching, three different strategies are proposed: vertical propagation, horizontal propagation, and probing. In the first case, advertisements (containing QoS information) are propagated up the hierarchy after each session. At each level the advertisement is either cached or the QoS value is updated as well as access frequency. Horizontal propagation involves sending the advertisement along the path to the client of the completed session. In the last case, if a client does not get an advertisement that contains QoS information, it sends a specific request for one that does. This forces intermediate nodes to bypass their cache if they do not have QoS information.
Chapter 6

Security

Security is an important issue in any service that is run over wide area networks, the internet especially. The sheer number of users ensures that attacks will be made. Malicious servers must be prevented from producing false advertisements which can be used to redirect users from legitimate sites, obstruct the use of service by flooding it with bogus service announcements, impersonate servers or make unauthorized access to services.

6.1 Sevice Location Protocol

SLP [23] uses authentication based on certificates issued by trusted authorities. Security features are not required in all communications, but all updates, advertisements and such do require it. Authentication Blocks are added to these messages and contain the certificate and the location of the authority. The block also contains information that allows the receiver to check for tampering. Authentication is only required in the local domain, so in the wide area case it is not possible to check agent identities. SLP does not provide any access control, every advertisement can
be read by any client. SDS provides a more complete security infrastructure.

6.2 (Secure) Service Discovery Service

The developers of SDS [6] have made security a principle issue in their design. SDS uses several techniques to achieve security within the system: authentication, capabilities, and encryption. The following describes how these are used.

Certificates are used to provide authentication within the system. This requires a trusted and available Certificate Authority. Authentication then insures that encrypted messages are secure. Authentication and encryption allows clients seek out only services located on trusted servers. Conversely, if a service wishes to limit who can view the announcement, SDS provides a mechanism that associates capabilities to announcements so that only those with the appropriate capability may learn about the service. Services contact a Capability Manager (securely) and give an access list. The Capability Manager then produces capabilities for each client in the list. Capabilities are secure because they are signed with the client’s public key.

Server Announcements: clients must be able to trust information from the discovery service. SDS achieves this by signing and timestamping announcements but they do not encrypt the announcement itself.

6.3 Separate Discovery and Access

If security and authentication are provided by the resource sharing system then any resource location service that wishes to include those resources must also maintain the same level of security. This does not mean the location service must share the same security scheme. However, the resource sharing system must rely on the loca-
tion service to authenticate users as some advertisements may not be public. The challenge of the service is to provide authentication even though advertised resources come from many different systems which all have different security schemes. Burdening the location service with all this authentication is not appealing. A simple though not ideal solution is to limit the service to public advertisements and only authenticate advertisers similarly to SLP. The service could have its own authentication scheme for accepting advertisements. Private advertisements are probably less likely to be widely accessed and could presumably be obtained directly from the given system.
Chapter 7

Conclusion

In response to the growth of the internet, resource location systems have needed to deal with more information, more requests, and more variety. With the advent of pervasive computing and mobile networks the task is becoming even more challenging. In order to meet this challenge we must use the best techniques and learn from past systems.

Of the many techniques and strategies discussed in this essay the following standout as apt candidates for an internet wide service location service: Hierarchies, XML, Advertise/Subscribe.

Scalability requires a hierarchy, and the associated search strategies. The amount of data is simply too large for other structures to function properly. Hierarchical search strategies have shown themselves to be the an appropriate compromise between speed and efficient use of resources.

Flexibility such as provided by XML will allow a system to interact with the many resource access standards now being developed such as Jini, Bluetooth, and Salutation, but also with old protocols. Its existing popularity and adoption by industry make it the best candidate for a universal naming scheme.
Advertise/Subscribe schemes are most likely to succeed in providing adequate performance and reliable information. They demand little of the system in terms of gathering and maintaining data. Yet, they provide arguably the most accurate information.

Issues such as security and quality of service in location service systems have not been dealt with in great detail by existing systems. In the absence of actual use it is hard to predict what will succeed in these areas.

Overall, these techniques will provide a system with its best chance to meet the needs of the future internet.
Bibliography


