Carmen: A Dynamic Service Discovery Architecture

Sergio Marti, Venky Krishnan
Mobile and Media Systems Laboratory
HP Laboratories Palo Alto

HPL-2002-257
September 16th , 2002*

E-mail: {sermarti, venky} @hpl.hp.com

ubiquitous computing, peer-to-peer, service discovery

As the growth of connected devices accelerates, it becomes increasingly necessary and difficult to locate useful services quickly, no matter where they may be hosted. Local service discovery protocols are limited to nearby services. Centralized directory servers often contain out-of-date information and are unaware of resources local to the consumer [4] [29]. A method of bridging the two is needed. We present a service discovery architecture that scales to global proportions, maintains up-to-date information on short-lived services, and allows any provider to input its own service descriptions. It is sufficiently flexible to work well as a statically arranged contextual hierarchy or as a dynamic peer-to-peer network.

Carmen: A Dynamic Service Discovery Architecture

Sergio Marti, Venky Krishnan
Hewlett-Packard Labs
{sermarti, venky}@hpl.hp.com

September 10, 2002

Abstract

As the growth of connected devices accelerates, it becomes increasingly necessary and difficult to locate useful services quickly, no matter where they may be hosted. Local service discovery protocols are limited to nearby services. Centralized directory servers often contain out-of-date information and are unaware of resources local to the consumer [4][29]. A method of bridging the two is needed. We present a service discovery architecture that scales to global proportions, maintains up-to-date information on short-lived services, and allows any provider to input its own service descriptions. It is sufficiently flexible to work well as a statically arranged contextual hierarchy or as a dynamic peer-to-peer network.

1 Introduction

The next few years will see a massive increase in the availability of powerful wireless mobile devices [11]. These devices may be able to provide different complimentary services, such as GPS, web access, or machine controls. Working together they can augment their functionality and provide more powerful tools to their users. However, their mobility constrains their ability to statically identify services. Therefore it will be essential to provide an automatic method for dynamic service discovery in a wide area network.

Current network nodes participating in service exchanges can be classified as two types: dynamic hosts and backend servers. Backend servers are characterized by static network addresses and very little downtime. These tend to be web service providers maintained by large organizations, or consumers in business-to-business applications. Dynamic hosts include both mobile devices and typical PCs that are turned off or disconnected when not in use. These machines have traditionally been only service consumers. Peer-to-peer application networks have changed this by allowing dynamic hosts to connect together and share information and resources, such as files, with each other thus acting as both service providers and consumers. This trend will continue into the future as consumer devices become faster and better connected [44]. We have designed the Carmen dynamic service discovery network to enable a very large number of heterogeneous nodes, both dynamic and static, to offer and search for an unlimited number of services and information.

A key underlying assumption in this research is that clients (dynamic nodes) use wireless network interfaces like cellular, 802.11, or Bluetooth for network connectivity. In such environments, the clients access servers in the Internet by going through a static node – an access point or edge server. The access point acts as a gateway for the client. HotSpot servers [15], a service delivery environment for nomadic users, define one such system. Carmen is the service discovery mechanism for such environments and was developed as part of the HotSpot research.

The Carmen architecture can be described as an application network with three types of nodes (that mirror the network structure described above):

1. *Carmen client*: an end user agent that typically requests resources/services.
2. *Carmen proxy*: provides the link between clients and services. This mimics the functionality of an access point.
3. *Carmen service provider*: the service provider.

Throughout the paper we describe Carmen as a platform and mechanism for service discovery between consumer nodes and service providers. One function of such a network is search capabilities across a vast number of domains. A highly distributed Carmen network could provide the functionality of search engines but with the freshness of information of peer-to-peer networks. In fact, though the Carmen architecture is most efficient when manually configured into contextual hierarchies, it can also function well as a collection of autonomous nodes that dynamically build a tree hierarchy. In this manner it does resemble a structured hybrid peer-to-peer network.

Clients connect to the proxies as leaves of the tree. They advertise the services they make available, and/or query for services they need. The system is targeted towards a semi-static network topology. End hosts may be very mobile and connect and disconnect frequently, but the proxies will mostly be located at static servers throughout the Internet. To reduce traffic and improve search results we use context information to construct the proxy tree and impose service domains on the proxies. Maintaining sufficient reliability and availability in such a large network requires several fault tolerance and replication techniques, which we will discuss.

Section 2 illustrates some of the application scenarios that would use dynamic wide-area service discovery. The next three sections describe the main contributes of this research. Section 3 covers the Carmen network architecture. Section 4 deals with how service descriptions are used. Contextual hierarchies, our method for enabling efficient searches, are described in Section 5. The current implementation status is covered in Section 6. Section 7 discusses scalability, fault tolerance, and security issues in the network. Finally, we summarize and conclude in Section 8.

2 Motivation

The growth of ubiquitous computing will see the spread of autonomous agents and distributed systems in devices surrounding us. To be effective these devices will need to interact and leverage each other's services in order to provide the user with a productive and/or entertaining experience. But in order to work together there needs to be a mechanism for devices to find each other based on the complementary services they require.

The Carmen project began as an effort to design a powerful service discovery protocol for "smart" access points capable of delivering a variety of services to mobile hosts [15]. Because of our focus on mobile users, we are targeting an environment of both mobile and static end user devices connected to the Internet through static access points. We wish to use the static nature of the core network to facilitate search and service discovery for dynamic users. Though originally targeted to pervasive computing, Carmen can be deployed in a variety of environments and can bridge the gap between local dynamic service discovery and global static search.

In this section we look at two possible future applications in such an environment that would need such a discovery mechanism. We then look at the specific requirements of the discovery system, which we hold as the goals of the Carmen system. Finally, we look at how currently deployed technologies must evolve to meet the needs of future applications.

2.1 Scenarios

The following two scenarios demonstrate applications that use global dynamic search capabilities to share and locate information globally, even among mobile hosts. These are only two of the various services we have envisioned, all using one global distributed service discovery system.

2.1.1 Comparison-Shopping

Mike is driving across the country when his car breaks down a few miles from the nearest town. He pulls out his personal digital assistant with wireless support and connects to the nearest access point. Using his service locator application he requests information on tow trucks within the city limits and gets back a list of replies including their cost of tow and estimated time of arrival. He selects the closest one. While the tow truck is on the way he searches for nearby mechanics and requests the soonest appointment time available. He picks Big Bob's Garage, which replies that it can service his car in an hour. By now the tow truck has arrived and Mike directs it to Big Bob's place.

On the way to the garage, Mike realizes it's 1pm, he's really hungry, and in the mood for pasta. So using his PDA he requests a list of Italian restaurants within a couple of blocks of the garage with wait times of less than 15 minutes. He selects Don Giovanni and makes a reservation for 1:15. Once the car is safely delivered to Big Bob, Mike heads over to the restaurant to eat.

The preceding scenario illustrates a comparison-shopping service that allows users to easily request service information matching a wide variety of criteria, including geographic and temporal locality. To provide real-time information the service providers themselves must be queried directly if their static characteristics match the request. The HP Cooltown initiative reflects scenarios like the one above [25].

2.1.2 B2B

ACME Supplies makes office equipment using plastic widgets. A large order arrives and their supply management system realizes there aren't enough plastic widgets in stock to complete the order. So a request is sent to plastic widget manufacturers, asking which of them can deliver the necessary amount of widgets within 2 days. Widgetron Inc., a plastic widget manufacturer, receives the request at their Sales department. Their system, in turn, queries their corporate warehouses to see if any warehouse has sufficient widgets to meet the order and deliver them to ACME on time. The inventory system at warehouse 21 replies that they have enough widgets in their inventory and are located close enough to ACME to deliver them in a day. The Sales system records the information, replies to ACME that it can meet their needs, and provides a URI to complete the order process.

When the order is processed and delivered to the manager at warehouse 21, she checks which packaging team at the warehouse is currently available to handle the order and forwards it to them. The next afternoon a large shipment of plastic widgets arrives at ACME Supplies in time to meet their order deadline.

2.2 Goals

The scenarios above illustrate certain requirements the end applications would need from the discovery protocol. Below we discuss each of these requirements, and see if and how they are handled today.

2.2.1 Dynamic

In a network composed primarily of dynamic hosts offering services it is vital that the service discovery mechanism use fresh up-to-date information. Due to node mobility and increased interaction, advertised services may be available for the span of minutes, not months. And as the number of service providers increases, a centralized directory will not be able to maintain fresh and accurate information about the services and their characteristics. We believe a distributed directory service is needed to handle the enormous number of updates required in order to have fresh service location information. Currently used global service directories, such as the Universal Description, Discovery and Integration of Business for Web (UDDI) [45], are centralized directories geared only towards static persistent services to backend servers. Such solutions will not adapt to the rise of dynamic host services.

For some providers, though the types of services offered may rarely change, the characteristics of the current service may be constantly changing. Propagating such dynamic service attributes with service advertisements greatly increases network traffic, message processing time, and server state, while not giving any benefits to query processing since the attribute information may be stale. By routing queries to service providers based solely on service name and allowing the provider to process the service attribute requests gives consumers freshest responses possible, while greatly decreasing the cost of advertisement propagation, but with an increase in query propagation.

2.2.2 Scalable

Though there are several standardized service location protocols for use within a local area network, little research has been done on providing service discovery across many administrational domains approaching a global solution to service discovery. An architecture capable of spanning the range of local service discovery to global search capabilities would have to be highly scalable. The need for such a scalable service discovery protocol suggests using a hierarchical organization similar to the Domain Name System (DNS), a global service for mapping human readable names to internet addresses [35][36].

2.2.3 Context-based routing

Current peer-to-peer networks of dynamic nodes suffer from large traffic overhead for disseminating service information and propagating queries throughout the entire network. If some nodes are known to be relatively static, they can be used to organize the service information and coordinate query propagation. By putting some contextual knowledge related to the service queries into the proxies it is possible to make more intelligent query and advertisement routing decisions. This would decrease network traffic while maintaining or improving query results as compared to the current blind flood routing methods of peer-to-peer networks. Methods of constructing peer-to-peer topologies based on connecting nodes with associated resource categories have been researched [10][38].

2.2.4 Access Control

Disregarding the scalability issue with a centralized service discovery architecture, there is one major functional limitation of such a system. A single directory server would prohibit service providers from confidently limiting access to their service announcements as they see fit. Most service providers will want to offer a different set of services to different entities based on their relationship to those entities. Also, many services are only appropriate within a domain and knowledge or access to those services outside of the domain must be prohibited.

For example, in the second scenario, Widgetron Inc. handles requests for plastic widgets. Their warehouses answer queries regarding their current inventory, but only if the queries come from within the corporation. Therefore, though Widgetron may advertise plastic widgets globally, warehouse services should not be advertised outside of the company, nor should requests for such services originating from outside be accepted. Similarly local packaging services at the warehouse do not need to be advertised to the other departments of the corporation.

This implies the need of a multi-tiered access control system. A completely centralized service discovery approach would either offer no fine-grain service access control, or require service providers to trust the central directory with managing access to all of their services.

Today, multi-level service lookup is commonly handled by separate directory servers at different levels of an organization. Global services are advertised through UDDI [45]. Corporate services are accessed via LDAP [49]. Local services may be discovered using Universal Plug and Play (UPnP) [20][33] or a Service Location Protocol (SLP) [23][24] directory agent to search beyond a subnet. This requires clients to know which server to contact for each type of service request or advertisement, adding extra management overhead beyond the required access control at the servers. We believe it is preferable to remove directory server configuration information from the end clients and embed it in the service location network, while maintaining access control restrictions at the servers.

2.2.5 Flexible

The architecture must allow service providers to advertise services that may not be registered with a central authority, yet support interoperability between service consuming applications and service providers. New services should provide a service description template specifying the attribute names, types, and values related to the service. If the service descriptions adhere to a specified schema then applications can present human users with service-specific query forms and browsers for any service, without having any a priori knowledge of the service or its attributes and syntax.

It is equally important for a service discovery mechanism to facilitate programmatic service discovery performed by an application without the user's input. For widely accessed services a central authority to standardize services and their descriptions would allow interoperability between various consumers and providers. This is the method adopted by SLP [23] and UDDI [45]. Service standardization is not necessary when both the end point applications are developed by the same entity, as is the case with most low profile software.

Many of the proposed and available service discovery protocols accomplish some of the goals we have set forth, but none accomplish all the goals at an acceptable level for the applications and scenarios we have in mind for Carmen.

2.3 Current Technologies

Perhaps the most well known type of discovery service used is the web search engine. Search engines, such as Google [21] and Altavista [3], create massive indices of the millions of web pages they crawl; allowing users to query for pages relevant to their interests using pattern matching. Search engines have progressed from only cataloging web pages to allowing searches on images, Usenet newsgroups, media files, and more. To assist in searching for information on specific topics, sites such as Yahoo! [51] provide directories of all web sites sorted into subject-related categories and sub-categories. This allows users to use the directory service like a Yellow Pages for services, or to narrow the subject scope of the index for their pattern match search.

Though search engines and directory services work well with static web services and information, they are not suitable for searching for dynamic services. Because of the enormous amount of information, web search engines cannot maintain a fresh index of all websites. Though the most popular sites may be crawled more often, the average website index may be months old [4][29]. Directories with months-old stale information are not going to be suitable for locating web services with lifespans of minutes or hours.

By distributing the index of content and services across many different servers, the time and cost of maintaining the index in each server's scope is greatly reduced. Web services can register and update the directory servers handling their most specific categories. A large distributed directory service must determine where queries are sent and how to efficiently route them.

The two largest peer-to-peer file-sharing networks, Gnutella [19] and FastTrack [17], have millions of users sharing terabytes of data. But peer-to-peer networks suffer from two deficiencies. First, the traffic overhead of propagating file queries to all nodes is enormous. And second, the scope of the queries is too constrained, limited only to files, and not sufficiently flexible to allow users to create file categories to search within.

Existing peer-to-peer networks and research in the area have tackled the first problem through various means such as biased random walks [1], incremental radius [53], smart replication [29], and distributed hash tables (DHT) [39][41][42][54]. One improvement commonly used by public hybrid peer-to-peer networks is super-nodes [17][26][52]. These are specific nodes shown to be reliable and persistent over a period of time that are chosen to index the information cached at their neighbors. Queries then need only be routed among supernodes to check for matches. Therefore supernodes act as a higher tier in the network, above most of the nodes. This two-tier hierarchy could be expanded to more levels and further decrease query traffic.

3 Carmen Network Architecture

Carmen's basic operation involves service providers advertising the types of services they offer to a local proxy, which propagates the advertisement throughout the Carmen network. Consumer nodes can request a specific service with certain attribute constraints or request a list of available services whose names match a given regular expression. These requests are sent to the consumer's local Carmen proxy, which forwards the request through the network. Queries reach the service providers advertising the requested service. The providers check if they can meet the service demands specified in the query attributes and reply directly to the consumer if they match. The providers send the contact address or URL of the service and any additional service information. The consumer may then choose a service provider from the replies it received and negotiate directly with that provider to access its service.

In order to minimize the amount of service advertisements that are propagated up the hierarchy only a service's name, short description, and template pointer (URL) are advertised and propagated through the network. Service attribute information is not included in advertisements. Though this requires that service queries reach the service providers, this decreases the service advertisement size and improves the freshness and semantic power of query processing by having the service provider handle it.

Carmen allows nodes to offer and request new services by creating simple service descriptions in the Extensible Markup Language (XML) [48] conforming to a general schema or DTD. These service description templates may be standardized by an authorized organization, or simply be promoted by their developers. Service descriptions are described in Section 4.

3.1 Topology

To support a very large number of consumers and service providers with relatively static nodes that constitute a distributed lookup service we arrange the nodes in a hierarchical tree topology. This structure allows for large scalability and facilitates our context-based routing schemes. For example, the Domain Name System operates as a hierarchical structure handling billions of lookup requests every day [31].

The way the tree structure is used in Carmen is as follows: When a Carmen proxy receives a message with service advertisements, from a Carmen agent or from a child proxy, it records the source node and adds the received advertisements to its list of advertised services. It then forwards its expanded list of services up to its parent, which repeats the process. Each proxy maintains, for each child, the services it advertises, and, for each service, the child nodes that advertise it.

When a proxy receives a query message directly from a consumer or one of its child proxies it propagates the message up to its parent. It also checks to see if other child proxy advertises the service requested. If so it forwards the query to it. When a proxy receives a query message from its parent, it checks its children and forwards the query to any child proxy or service provider that advertises the service. The service providers process the query and reply directly to the proxy local to the requesting consumer on a positive match.

Advertisement and query messages contain hop counts, which are decremented each node up the tree towards the root and not propagated any further when the count reaches 0. This limits traffic and exploits the contextual information embedded in the tree structure, and is further discussed below.

A one-tier network would consist of a Carmen proxy connected to one or more consumers and service providers. The providers advertise their services to the proxy. The consumers send service queries to the proxy, which then forwards them the any provider advertising that service. A one-tier Carmen network is similar to many other service discovery protocols. The Carmen proxy is equivalent to the SLP Directory Agent [23] or the Jini lookup server [43]. In fact it is possible to merge currently deployed service discovery protocols onto the leaves of a wide area Carmen network.

3.2 Bootstrapping (Joining the network)

When a Carmen client starts, it must locate the nearest Carmen proxy. The address may be manually supplied by the user to the application or determined automatically from the network.

Manual configuration would work for static computers and devices. It would also be the most likely choice for configuring the proxies themselves, allowing the network administrators to determine the structure of the Carmen network within their domain.

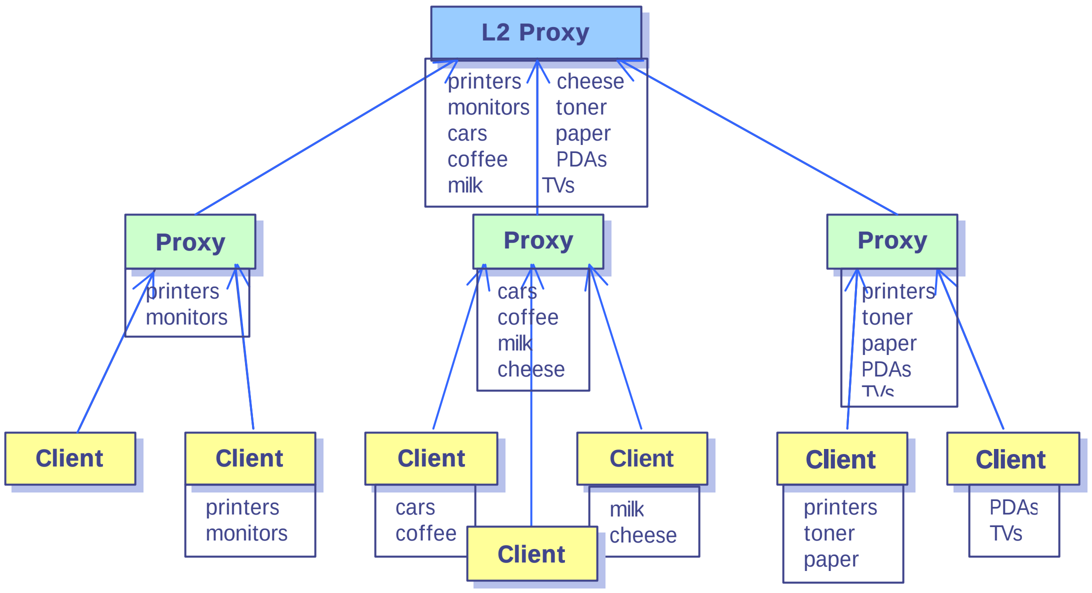


Figure 1: Illustration of service advertisement propagation and aggregation

But for most end users, their devices should dynamically locate the closest proxy. The address of the local proxy could be provided to a connected node through an extension to the DHCP protocol. DHCP Option 78 is already used to distribute the addresses of SLP Directory Agents. A similar technique would work well for Carmen. Another solution would be for nodes to use a well-known universal name that is locally mapped to the local Carmen proxy. For example *http://local-carmen-proxy/* could be resolved by the local DNS server to the address of the local Carmen proxy.

If no infrastructure support is available to provide the address of the nearest Carmen proxy then the joining node can use multicast discovery. HELLO messages are sent to a well-known Carmen multicast address with increasing TTLs. Any proxy receiving a HELLO message responds with its own address and information. The new node chooses a proxy from the responses and connects to it. Once contact with a proxy is established a node registers its service advertisements with the proxy.

Carmen proxies also support dynamic network configuration. This allows users to build an ad hoc Carmen network when disconnected from the Internet. Proxies use multicast advertisements to locate one another. If one of the proxies belongs to an established network of two or more nodes, it invites the new arrival to join as a sibling by giving it its parent's address. The new node then contacts the parent proxy, which can accept it as a child or reject it if the new node is not authorized to join. If a parent proxy has too many children, it divides its children into groups, and elects a leader node for each group. Each group's members now connect to the group leader, which then connects to the parent node. In this manner the network slowly grows the tree as the number of nodes grow.

3.3 Protocol Messages

Carmen messages are broken down into three types: service advertisements or updates, queries, and network management messages. Below each type is discussed in detail.

3.3.1 Advertisements

Service advertisement messages are used to propagate service information. A message can contain either a complete list of services advertised by a node and its descendants, or it can contain only the changes in its service list since the last advertisement was sent. A flag in the message specifies whether it is an absolute or delta list. For each service its name, description template URL, and optional short description is provided. In addition a hop count is added specifying how far up the tree this service advertisement should be propagated. This reduces advertisement traffic higher up the tree and decreases the queries handled by local service providers.

When a node receives an advertisement message it updates the list of services offered by the sending child and prepares an updated service list to send to its parent.

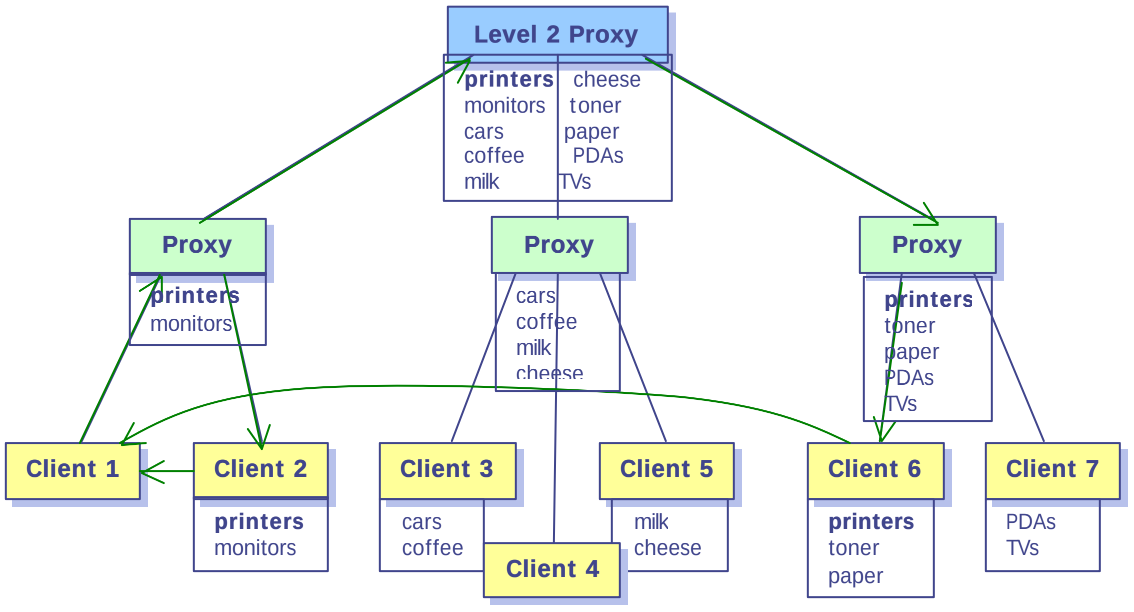


Figure 2: Illustration of service query propagation. Client 1 sends a request for service "**printers**" to its local Carmen Proxy.

When a client leaves the network its proxy removes the services advertised only by that client and constructs a new list of service advertisements from its remaining clients. The proxy creates a delta of the removed services and propagates this delta to its parent node, which repeats the process. This allows service advertisements to be cleanly revoked when clients disconnect from the network.

3.3.2 Queries

Query messages are used to forward requests from consumers through the Carmen Network. A query consists of a service name and a list of zero or more attribute specifications. An attribute is specified with its name, its type (string, integer, Boolean, etc), and the acceptable values for the consumer wishing to use the service. The attributes in a service query should be a subset of the attributes specified in the service's XML template. A pointer to the service template is provided with every service description. In addition the Carmen client API provides a special function to return the attributes for a given service.

Each query message contains a hopcount specifying the distance up the Carmen tree the request should be allowed to propagate. This allows for more accurate searches within a specific context (e.g. local services in a geographic context) and also reduces the total query traffic.

While advertisements are only updated on a periodic basis, queries must be propagated on demand to lower service discovery latency. This means request traffic is not automatically bounded. Therefore, several techniques can be used to limit the network cost of a large number of queries, especially at the higher levels of the Carmen tree. Such methods include child proxy polling, query batching, and simply dropping queries at overloaded proxies. Some of these techniques are discussed in Section 7.1.

If a service provider wishes to reply to a consumer's query, it notifies its local proxy, which sends a reply message directly back to the consumer's local proxy. The message contains one or more ServiceProvider tags. Each tag specifies the name and address of a service provider node and the method to contact it to negotiate service usage. It can also include additional service attributes to help the consumer decide between several responding providers.

3.3.3 Control Messages

Control messages deal with creating and maintaining the Carmen network structure. When a Carmen node wishes to join the Carmen network it sends a HELLO message to the address of an established Carmen proxy specified manually in its configuration or retrieved automatically. The proxy responds with its own HELLO message accepting it as a new child or rejecting it.

Control messages are also used to maintain local network topology information among proxies and to detect and repair node failures. Each parent node periodically sends its children keepalive messages with a list of its parents and its child nodes. The information is used by the children to elect a new parent node, should the existing parent fail. This is discussed in more detail below in the section on fault tolerance.

Child nodes do not need to generate separate keepalive messages since such information is piggybacked on service advertisement messages. They use them to detect and recover from parent node failure.

3.4 Interoperability

The architecture and protocol presented here is used for all communication between proxies, consumers, and providers. This does not need to be the case. It is likely that real-world deployments would use other service discovery protocols such as SLP, UPNP, or Jini in the local area. A Carmen proxy could easily encapsulate local service information into the Carmen format and propagate the information into a Carmen wide-area network. This gives the advantage of allowing different organizations to use whatever service discovery protocols are most efficient for their local area needs and use Carmen to connect the heterogeneous local protocols into one large global service discovery network.

4 Service Descriptions

All service advertisements are required to include a URL to a valid service description template. This template is an XML document that conforms to the Carmen service description DTD. As an example, a service description for network printers is provided in Appendix A.

4.1 Standardized Templates

The use of a standard service description DTD allows a Carmen-aware application to fetch any service's description template, parse it, present the choices to the user, and construct a valid service request based on the user's input.

Though any service provider can create their own service name and description and advertise it in the Carmen network, for consumer applications to effectively discover and access services through Carmen, they must have a priori knowledge of the services and their attributes. Therefore, automated consumer/provider interoperability requires an officially recognized service template repository, which provides a one-to-one mapping between a service name and a "well-known" standard template. Service providers may submit their service descriptions with this central authority. By using standardized service descriptions different consumers and service providers can negotiate service agreements. This is similar to the SLP approach of registering document templates with the Internet Assigned Numbers Authority (IANA) [23] or the UPnP Forum standardizing service schema and templates [46].

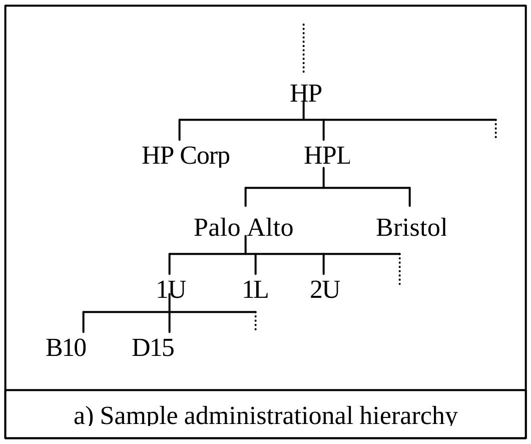
Global attribute-based service discovery is only viable if service descriptions and their templates are standardized. For Carmen to be accepted by the public its protocols and templates must be approved by a standards committee.

4.2 Ad Hoc Templates

Carmen also permits non-standard *ad hoc* services to be advertised by providers. This allows quick deployment of new services or services that will require user attribute input. A consumer application, though not familiar with a service's description, can request its template and present the attributes to a user for them to choose the search criteria. In fact in the current implementation all proxies have a web front-end (as well as an API front-end) allowing anyone to connect using a web browser, request list of services, request service information, and make attribute-specific queries. The proxy then returns a list of responding service providers with their contact addresses.

As an example of a human interactive application of Carmen we will use the example of a person shopping for a car online using a Carmen-aware shopping application to look for services related to cars. The network will return advertised services whose short descriptions mention cars. The application presents these descriptions to the user. The user selects a specific service type and the application fetches that service's template. The application then presents to the user the service attributes and possible values. The user selects the attributes they care about (automatic or standard, color, price range, etc) and leave the others blank. The application then queries the network for this service type with the specified attribute constraints. Online car sellers will receive the query, check their inventory for matches, and reply to the user with more detailed information about the matching cars and their online web address.

The issue with allowing any service provider to advertise any service name and description without having to be globally recognized deals with service name collision. Normally, when a proxy receives advertisements with the same service name from two children, it assumes they refer to the same service and therefore it only forwards one advertisement for the service. But two providers may advertise using the same service



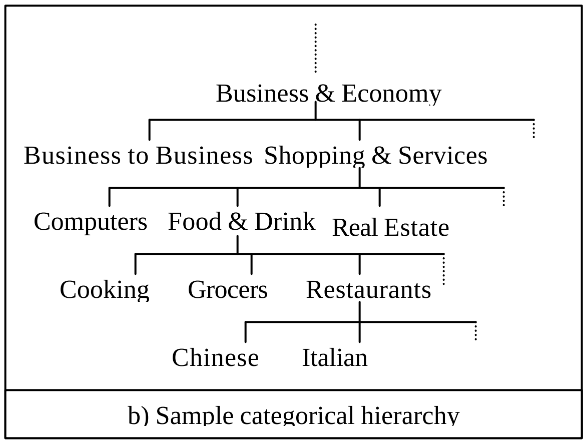


Figure 3: Two sample contextual hierarchies.

name but different service description templates. There are a few solutions to this problem. A proxy that receives two contradicting service advertisements should see if one uses a standardized template and accept only that one. A standardized template's URL would reference a valid XML document located at the central authority's well-known server. If neither service template is registered with the central authority then another criteria could be used to choose one over another, such as age (older or newer), reputation of the provider, or number of providers advertising a given template. In addition, a message could be sent back to the service provider informing them of the name collision. Another solution would be to associate a unique identifier to each service to distinguish the two within the Carmen network similar to namespaces in XML [48].

5 Contextual Hierarchies

One of the goals of Carmen is to use the hierarchical layout of the network to improve query efficiency. By assigning context roles to the proxies we focus their scope of interest. This limits the traffic they receive to traffic related to their area of specialization, allowing them to maintain more up-to-date information within their category and provide faster, more accurate response to consumers. To illustrate this point we shall use two different contextual hierarchy examples.

In most of the paper we have assumed a Carmen tree layout based on geography and administrational domains. This is one useful contextual hierarchy. As a query is propagated up the tree the scope of the proxies it visits increases. For example, the first proxy may be located at a wireless access point and be familiar with services within its range. The next proxy may know about all services available on the current floor. The next knows about services in the entire building, the next the whole campus, and so on up through the corporation to a global listing of public services that can be accessed by other consumers. If someone wishes to print something, they would want to use a printer located on their floor. Therefore they limit their query with a hop count of 2. Likewise, printers would only propagate advertisements to a hop count of 2. So queries and advertisements for local services remain near the bottom of the tree. Examples of globally announced services in this example may be B2B service descriptions similar to the UDDI.

Another example is a categorical hierarchy similar to the Yahoo! directory structure [51]. Each proxy maintains service information for services within its assigned category, such as Food & Drink. Each if its child proxies are in charge of a subcategory, like Cooking or Restaurants. Service providers connect directly to the most specific category proxy possible. With large categories multiple proxies may be needed. Such a structure optimizes service information and advertisement aggregation, since all providers of the same service will connect to the same proxy, or proxy group, so only one advertisement is propagated up the chain. By maintaining an easily accessible category to proxy map, consumers requests can be routed up the tree through the most likely category match, thus decreasing response latency. Use of hop count for queries can limit service type searches to providers within a category at any level of the hierarchy.

A deployment of a categorical hierarchy would have geographically near proxies as the leaf nodes of the network. These nodes would maintain a cached directory listing with mappings of categories to proxy addresses. The proxy would route any local requests or advertisement messages to the most appropriate address without the end user having to be involved.

5.1 Multiple Interleaved Hierarchies

The preceding examples demonstrate how limiting the contextual scope of the service information in the nodes of the Carmen network allows for more efficient and precise searches without limiting the power of the queries. There are many different useful contextual hierarchies. Each can be geared for different service and query categories. With a global Carmen network we are not limited to one contextual hierarchy. Each proxy may participate as a node in different contextual hierarchies. This is especially useful for public Carmen networks that provide service discovery to the whole world, such as the in the second example. If queries specify which contextual hierarchy benefits them, they can be filtered and routed through the network in that manner.

6 Implementation

We currently have a working Carmen proxy prototype implementation written in Java. We are using a multi-tiered hierarchy in our test environment. The proxies are manually configured with which proxy to connect to. Several sample service descriptions have been created and a web browser accessible front-end is built into every proxy allowing us to test user-driven request and response behavior.

An API has been developed and Java stub classes are being used to add service discovery functionality to applications. We have also written a prototype application for testing and demonstrating the uses of the Carmen network. This application allows users to dynamically create a shared group and easily do remote file search and access across all the peers in the group. Current application work is targeted at the trusted LAN environment, but further work will be done to improve authentication and functionality of the application for wide area usage.

Our choice to not distribute a service's attribute information in its advertisement was to limit the size of the service advertisements, take advantage of service description aggregation at higher levels, and allow highly dynamic service providers to process requests themselves. This solution can result in a large amount of query traffic both up and down the tree if certain service names are used for very popular or broad services. Providing some attribute information in service advertisements, at least for static attributes, may decrease overall Carmen traffic. This is one of the optimizations we hope to investigate in the future.

7 Discussion

This section covers design details to several issues that have not yet been implemented. Currently Carmen's fault tolerance protocol is not fully implemented. Nor is there currently support for multiple interleaved context-based hierarchies. More detailed analysis of the protocol must precede finalizing an implementation of these functions.

7.1 Scalability

A hierarchical tree structure organizes the network into manageable structure and reduces the total amount of traffic significantly compared to a flooding approach, but it still suffers from high strain on the upper nodes of such a large tree. Therefore, we propose several techniques for reducing the demand on high-level nodes.

At the higher levels nodes must handle the accumulated service advertisements and queries generated in their subtrees, which may be composed of millions of nodes. One solution is to divide the message processing among a group of proxies. We propose using service namespace division to alleviate highly loaded proxies. Each group member will be allocated a subset of the service namespace. For example, proxy 1 would process messages relating to services beginning with *a-f*, proxy 2 would handle *g-k*, and so on. Child proxies would forward service advertisements and queries to the corresponding proxy based on the service name. The group members could maintain statistics and dynamically reallocate the namespace as necessary to balance load based on metrics such as total services advertised per node, or queries per second. At the next higher level of the hierarchy it may be necessary to have more proxies in the group and more finely divide the services. This addressing method is similar to that used by certain DHTs [41][54]. The division does need not be done simply by first letter(s) of the service names, but any hashing method that maps identifiers to a uniform space.

Another proxy group method for increasing availability and reducing per proxy traffic is to multicast messages to a group of proxies instead of just one. The two extreme options here are: advertise one/query all, or inversely, advertise all/query one (though intermediate solutions are also viable [18]). In the former method service advertisements are sent to only one proxy in the group while service queries are sent to all. In the latter service advertisements are sent to all proxies in the group while queries are only sent to one. Since service queries cause much more traffic than the managed periodic service information updates, the latter approach would be preferable.

As mentioned above, both advertisements and queries contain hop count fields to limit the diameter of the discovery process. The end agents primarily use this to limit searches to local geographic or contextual reasons. Intermediate nodes may shorten the hop count further given semantic knowledge of the search context, or based on current network congestion.

If the query rate at a proxy becomes too great it may ask its child nodes to aggregate queries into single messages and periodically send it batch query messages. This will decrease the per query cost of connection maintenance and XML parsing. The switch from reactive to proactive query propagation could happen when a proxy passes a query rate threshold and return to reactive forwarding the rate becomes manageable. This technique introduces query buffers in the child nodes. A query buffer limit may be set at which point queries begin to be dropped, similar to IP routing. A control message may be sent back to the request originator when their message is dropped.

Periodic query batches will also increase total request/response latency. The farther up the tree a query is, the greater the chance there is an acceptable service provider in the subtree below it. Delaying the query propagation at the upper levels will increase service discovery latency for consumers who are searching for rare items or services, or wish to collect a very large number of results before choosing. In most cases of automated service discovery hidden from the user, the service searched for will either need to be relatively nearby in the Carmen network so search latency will be low.

As the size of a proxy's subtree grows the expected aggregation of service advertisements for the same service is expected to increase. Though the amount of advertisement aggregation will be less than using other methods, such as Bloom filters [8][13], it provides lossless aggregation and allows partial lookups on service names or descriptions.

7.2 Fault Tolerance

The problem with hierarchical networks is the increased single points of failure. Any intermediate Carmen node that fails will partition its entire subtree from the rest of the Carmen network, assuming no replication. Therefore it is important to provide a robust method to be able to quickly detect and repair node failures in the tree. In a highly distributed large network cross various administrational domains system managers will likely handle the failure of a high-level inter-domain node manually. But Carmen does provide for automatic failure recovery.

Both parent and child nodes send each other periodic management messages to inform each other of their current status. This is especially important for detecting intermediate node failure and repairing breaks in the tree. If a node fails it is the responsibility of its children to elect a new parent to replace it and connect to the failed nodes parent. In each control message a parent includes a list of its children and a list of its parent proxies. When a proxy goes down, the child nodes must agree on one of their nodes to take the place of the failed proxy. One simple solution is to automatically elect the first child on the children list provided by the failed node to replace it. The parent could order the list based on some metric it has about its children, such as reliability, bandwidth, or latency.

When node detects a connected node may have failed it sends it a HELLO message expecting an immediate reply. If no reply arrives it declares the node dead. When a child node fails, the parent node simply removes it from its list of children. When a parent node fails, any child that detects it contacts the first child on the latest children list it has received from the parent, notifies it that the parent node has failed and that it should take over. That node then contacts the other child nodes and the failed nodes parents, informing them that it is replacing the failed node. The other children now send full service lists to the promoted node so that it has a fresh list of all services advertised below it.

7.3 Security

Hierarchies effectively model administrational structures that the offered services may belong to. Different subtrees within the large global network constitute different organizations or companies. Within these subtrees smaller subtrees may correspond to departments. Having each subtree have a single root node, simplifies enforcing service discovery policies. Acting much like a firewall an administrational root node can be configured to allow only certain services to be advertised outside of its domain. Likewise, it can limit which types of service requests (and by who) are allowed into the domain.

Carmen does not inherently provide any security or authentication of consumers or services. It does not provide any protection for service discovery. The service providers and consumers themselves should negotiate any authentication or privacy needs when attempting to access the service. Within a protected domain, such as behind a firewall, access is often equated with authorization. In such situations security of resource discovery may not be necessary. Other services are publicly advertised and their locations are well known. Protection is needed at trust domain edges. One simple method is to place service and query propagation policy at administrational domain boundaries. For higher security other techniques are available.

Methods have been developed for securing service discovery in similar networks. Most notable is the work at Berkeley on securing their Service Discovery Service [13]. The methods developed there could be applied to the Carmen network but we see them as unnecessary for our target scenarios.

8 Conclusion

Our work on Carmen has given us insight into the requirements of a scalable, dynamic service discovery. We believe it is possible to effectively meet these requirements with a distributed architecture based on contextual hierarchies. The current prototype implementation of the Carmen proxy has proven sufficiently flexible to allow easy testing of different routing schemes and construction of applications that use the Carmen network to search for and share information in ways we had not originally envisioned. With continued development and focus on large-scale simulation using real data from our testbed, we hope to soon demonstrate its effectiveness as a system for locating services across the whole Internet.

Acknowledgements

The authors would like to thank Lada Adamic, John Barton, Vana Kalogeraki, Harumi Kuno, and Jeff Morgan for their helpful advice and comments on early drafts of this paper.

References

[1] L. Adamic et al. *Search in power law networks*. Physical Review, E 64 (2001), 46135-46143.

[2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan and J. Lilley. *The design and implementation of an intentional naming system*. 17th ACM Symposium on Operating Systems Principles. 1999.

[3] Altavista – The Search Company. www.altavista.com

[4] P. Bailey, N. Craswell and D. Hawking. *Dark matter on the Web*. In Poster Proceedings, 9th World-Wide Web Conference, 2000.

[5] M. Balazinska, H. Balakrishnan and D. Karger. *INS/Twine: A Scalable Peer-to-Peer Architecture for Intentional Resource Discovery.* Pervasive 2002 - International Conference on Pervasive Computing, Zurich, Switzerland, August 2002.

[6] J. Barton. Personal Communication. August 2002.

[7] C. Bettstetter, and C. Renner. *A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol*. Proc. 6th EUNICE Open European Summer School: Innovative Internet Applications (EUNICE'00), Twente, Netherlands, September 13-15, 2000.

[8] B. Bloom. *Space/Time Tradeoffs in Hash Coding with Allowable Errors*. Communications of the ACM, 13(7):422-426, July 1970.

[9] P. Castro et al. *Locating Application Data Across Service Discovery Domains*. Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking. 2001.

[10] E. Cohen et al. *Finding Interesting Associations without Support Pruning*. Technical Report, Computer Science Department, Stanford University, 2001.

[11] Computer Industry Almanac Inc. Press Release, March 21 2002. www.c-i-a.com/pr032102.htm.

[12] N. Craswell, P. Bailey and D. Hawking. *Server selection on the World Wide Web*. In Proceedings of the Fifth ACM Conference on Digital Libraries, pages 37-46, 2000.

[13] S. Czerwinski et al. *An Architecture for a Secure Service Discovery Service*. Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking. 1999.

[14] P. B. Danzig, K. Obraczka, and A. Kumar. *An analysis of wide-area name server traffic: a study of the Internet Domain Name System*. ACM SIGCOMM Computer Communication Review, v.22 n.4, p.281-292, Oct. 1992.

[15] D. Das et al. *HotSpot! – a service delivery environment for Nomadic Users System Architecture.* HP Labs Technical Report, HPL-2002-134, May 9 2002.

[16] K. F. Eustice, T. J. Lehman, A. Moralies, M. C. Munson, S. Edlund and M. Guillen. *A universal information appliance*. IBM Systems Journal, vol. 38, no. 4, 1999. www.research.ibm.com/journal/sj/384/eustice.html

[17] FastTrack. 2002. www.fasttrack.nu/

[18] D. K. Gifford. *Weighted voting for replicated data*. Proceedings of 7th ACM Symposium on Operating Systems Principles, pp. 150-62, December 1979.

[19] Gnutella. 2002. www.gnutella.com

[20] Y. Goland et al. *Simple Service Discovery Protocol/1.0*. IETF, Draft draft-cai-ssdp-v1-03, October 28 1999. http://www.upnp.org/download/draft_cai_ssdp_v1_03.txt

[21] Google. 2002. www.google.com

[22] S. Gribble et al. *The Ninja Architecture for Robust Internet-Scale Systems and Services*. Special Issue of Computer Networks on Pervasive Computing, 2000.

[23] E. Guttman. *Service Location Protocol: Automatic Discovery of IP Network Services*. IEEE Internet Computing, vol. 3, no. 4, pp. 71-80, 1999.

[24] E. Guttman, C. Perkins, J. Veizades, and M. Day. *Service Location Protocol, Version 2*. IETF, RFC 2608, June 1999.

[25] Hewlett-Packard Company. *cooltown*. August 2002. cooltown.hp.com.

[26] V. Kalogeraki. Personal Communication. August 2002.

[27] KaZaA. 2002. www.kazaa.com.

[28] Butler W. Lampson, *Designing a global name service*. Proceedings of the fifth annual ACM symposium on Principles of distributed computing, p.1-10, August 11-13, 1986, Calgary, Alberta, Canada

[29] S. Lawrence and C. Lee Giles. *Accessibility of information on the web*. Nature, 400:107-109, July 1999.

[30] C. Lv and P. Cao and E. Cohen and E. Felten and X. Li and S. Shenker. *Search and replication in unstructured peer-to-peer networks*. Proc. 2002 ACM SIGMETRICS, 2002.

[31] C. D. Marsan. *Verisign CEO talks about strategy, integration*. Network World, p. 18, April 15 2002.

[32] R. E. McGrath. *Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing*. Department of Computer Science University of Illinois Urbana-Champaign, Urbana UIUCDCS-R-99-2132, March 25 2000.

[33] Microsoft Corporation. *Understanding Universal Plug and Play: A White Paper*. June 2000.

[34] D. Milojicic et al. *Peer-to-Peer Computing*. HP Labs Technical Report, HPL-2002-57, March 8 2002. www.hpl.hp.com/techreports/2002/HPL-2002-57.html

[35] P. Mockapetris. *Domain names – concepts and facilities*. RFC 1034, ISI, Nov. 1987.

[36] P. Mockapetris and K. J. Dunlap. *Development of the domain name system*. Symposium Proceedings on Communications Architectures and Protocols, 1988.

[37] B. Oki, M. Pfluegl, A. Sigel, and D. Skeen. *The Information Bus – An Architecture for Extensible Distributed Systems*. Proceedings for the 14th Symposium on Operating System Principles, 1993.

[38] M. K. Ramanathan, V. Kalogeraki, and J. Pruyne. *Finding Good Peers in Peer-to-Peer Networks*. HP Labs Technical Report, HPL-2001-271, Oct 23 2001.

[39] S. Ratnasamy et al. *A scalable content-addressable network*. In Proc. Of the ACM SIGCOMM Conference, San Diego, CA, 2001.

[40] J. Rosenberg, H. Schulzrinne and B. Suter. *Wide area network service location*. IETF Internet Draft, November 1997.

[41] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In IFIP/ACM Middleware, Heidelberg, Germany, 2001.

[42] I. Stoica et al. *Chord: A scalable peer-to-peer lookup service for Internet applications*. In Proc. Of the ACM SIGCOMM Conference, San Diego, CA, 2001.

[43] Sun Microsystems. *Technical White Paper: Jini Architectural Overview*.1999. www.sun.com/jini/.

[44] B. Thomas. *IT Trends: Global Mobile Wireless Services 2002 to 2005*. Giga Information Group, Inc., June 27 2002.

[45] Universal Description, Discovery and Integration (UDDI) Project. www.uddi.org.

[46] Universal Plug and Play Forum. 1999. www.upnp.org.

[47] S. Vaghani et al. *Infospaces: A Large-Scale Content Classification and Dissemination Network*. April 2001. www-db.stanford.edu/~svaghani/projects/infospaces/InfoSpaces-paper.pdf

[48] W3C. Extensible Markup Language (XML) Specification 1.0. www.w3.org/TR/2000/REC-xml-20001006.

[49] M. Wahl, T. Howes, and S. Kille. *Lightweight Directory Access Protocol (v3)*. IETF RFC 2251, December 1997.

[50] S. Waterhouse. *Jxta search: Distributed search for distributed networks*. Sun Microsystems, Inc., 2001.

[51] Yahoo!. 2002. www.yahoo.com

[52] B. Yang and H. Garcia-Molina. *Comparing Hybrid Peer-to-Peer Systems*. in Proc. of the 27th International Conference on Very Large Data Bases, September 2001.

[53] B. Yang and H. Garcia-Molina. *Improving Search in Peer-to-peer Networks*. ICDCS, 2002.

[54] B. Zhao, J. Kubiatowicz, and A. Joseph. *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*. Technical Report UCB/CSD-01-1141, Computer Science Division, U. C. Berkeley, April 2001.

Appendix A: Sample Service Description (Network Print Service)

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet
  href="http://weblab.hpl.hp.com/~carmen/xml/services/carmen_service.xsl"
  type="text/xsl"
?>

<!DOCTYPE Carmen:Service SYSTEM
    "http://weblab.hpl.hp.com/~carmen/xml/dtd/carmen_service.dtd">

<!-- XML definition of a print service query form -->

<Carmen:Service xmlns:Carmen="http://weblab.hpl.hp.com/~carmen/"
                name="print"
>
  <Carmen:Description>
    Allow access to printing services
  </Carmen:Description>

  <Carmen:Attribute name="Quality" type="String">
    <Carmen:Choices>
      <Carmen:Choice>High</Carmen:Choice>
      <Carmen:Choice>Medium</Carmen:Choice>
      <Carmen:Choice>Low</Carmen:Choice>
    </Carmen:Choices>
  </Carmen:Attribute>

  <Carmen:Attribute name="Color Depth" type="String">
    <Carmen:Choices>
      <Carmen:Choice>B/W</Carmen:Choice>
      <Carmen:Choice>Grayscale</Carmen:Choice>
      <Carmen:Choice>Line Art</Carmen:Choice>
      <Carmen:Choice>Photorealistic</Carmen:Choice>
    </Carmen:Choices>
  </Carmen:Attribute>

  <Carmen:Attribute name="Paper Size" type="String">
    <Carmen:Choices>
      <Carmen:Choice>Letter</Carmen:Choice>
      <Carmen:Choice>Legal</Carmen:Choice>
      <Carmen:Choice>Envelope</Carmen:Choice>
    </Carmen:Choices>
  </Carmen:Attribute>

  <Carmen:Attribute name="DPI" type="Integer" />

</Carmen:Service>
```

Appendix B: Carmen Message Specification

This DTD describes the currently used message format in the Carmen implementation.

```
<?xml version='1.0' encoding='utf-8'?>
<!-- Basic DTD for Carmen messages -->

<!ELEMENT Carmen:Message (Carmen:ControlMessage|Carmen:QueryMessage|Carmen:InfoMessage)>
<!ATTLIST Carmen:Message
    xmlns:Carmen CDATA #REQUIRED
    protocol     CDATA #REQUIRED
    id           CDATA #REQUIRED
    ttl          CDATA #REQUIRED
>
<!ELEMENT Carmen:ControlMessage
(Carmen:ControlHello|Carmen:ControlRequest|Carmen:ControlReply)>
<!ATTLIST Carmen:ControlMessage
    type CDATA #IMPLIED
>
<!ELEMENT Carmen:QueryMessage (Carmen:ServiceRequest|Carmen:ServiceResponse)>
<!ATTLIST Carmen:QueryMessage
    type CDATA #IMPLIED
>
<!ELEMENT Carmen:InfoMessage (Carmen:ServiceList)>
<!ATTLIST Carmen:InfoMessage
    type CDATA #IMPLIED
>
<!ELEMENT Carmen:ServiceList (Carmen:ServiceOffer*)>


<!ELEMENT Carmen:ControlHello (Carmen:SourceNode)>
<!ELEMENT Carmen:ControlRequest (Carmen:SourceNode, Carmen:ParentNodes,
                                 Carmen:ChildNodes)>
<!ELEMENT Carmen:ParentNodes (Carmen:Node*)>
<!ELEMENT Carmen:ChildNodes (Carmen:Node*)>

<!ELEMENT Carmen:ControlReply (Carmen:SourceNode)>


<!ELEMENT Carmen:ServiceRequest (Carmen:SourceNode, Carmen:Attributes,
                                 Carmen:LastNode)>
<!ATTLIST Carmen:ServiceRequest
    name CDATA #REQUIRED
    form CDATA #REQUIRED
>

<!ELEMENT Carmen:ServiceResponse (Carmen:SourceNode, Carmen:ServiceProvider+)>
<!ATTLIST Carmen:ServiceResponse
    name CDATA #REQUIRED
    form CDATA #REQUIRED
>

<!ELEMENT Carmen:SourceNode (Carmen:Node)>
<!ELEMENT Carmen:LastNode (Carmen:Node)>

<!ELEMENT Carmen:Node (Carmen:AddedServices?, Carmen:RemovedServices?)>
<!ATTLIST Carmen:Node
    address  CDATA #REQUIRED
    protocol CDATA #REQUIRED
    type     CDATA #REQUIRED
>

<!ELEMENT Carmen:AddedServices (Carmen:ServiceOffer*)>
<!ATTLIST Carmen:AddedServices
    type  (delta|full) #IMPLIED
>
<!ELEMENT Carmen:RemovedServices (Carmen:ServiceOffer*)>
```

```
<!ELEMENT Carmen:ServiceOffer (#PCDATA)>
<!ATTLIST Carmen:ServiceOffer
    name  CDATA #REQUIRED
    form  CDATA #REQUIRED
>

<!ELEMENT Carmen:ServiceProvider (Carmen:Attributes)>
<!ATTLIST Carmen:ServiceProvider
    address  CDATA #REQUIRED
    protocol CDATA #REQUIRED
    api      CDATA #REQUIRED
>

<!ELEMENT Carmen:Attributes (Carmen:Attribute*)>

<!ELEMENT Carmen:Attribute (#PCDATA)>
<!ATTLIST Carmen:Attribute
    name  CDATA #REQUIRED
    type  CDATA #REQUIRED
    comp  (Equal|Less|Greater|LessOrEqual|GreaterOrEqual) #REQUIRED
    value CDATA #REQUIRED
>

<!ELEMENT Carmen:Address (#PCDATA)>
<!ELEMENT Carmen:Protocol (#PCDATA)>
```

Appendix C: Consumer/Service Provider API

This is the Java API used by Carmen clients to advertise and query services from the network. It initializes the Carmen Stub classes which communicate with a specified proxy.

```
package com.hp.carmen.api;

import java.util.*;

/**
    This interface defines the methods consumers and service providers
    should use to interact with a Carmen proxy using a stub.
**/

public interface CarmenAPI {

    // Stops accepting Carmen-related messages
    public void stop();

    // Returns a list of serviceNames and their corresponding description URLs
    // given a regular expression to search for
    public int getServiceList(String regexp, List serviceNames,
                              List serviceForms)
       throws CarmenException;

    // Returns a list of ServiceAttributes for a given service
    public List getServiceInfo(String serviceName) throws CarmenException;

    // Initiates a search for the given query with the given ServiceAttributes.
    // Asynchronous: A ResponseHandler callback object must be provided.
    public int doServiceQuery(String serviceName, Vector attrs,
                              ResponseHandler rh)
       throws CarmenException;

    // Begin advertising a service with the given service description URL
    // Must provide a RequestHandler callback object to handle requests
    public boolean offerService(String serviceName, String serviceForm,
                               RequestHandler rh)
       throws CarmenException;

    // Send out query for service with given attributes.
    // Return strings containing address API form URL pairs

    public boolean removeServiceOffer(String serviceName)
       throws CarmenException;

}
```

==

```
package com.hp.carmen.api;
import java.util.*;
import com.hp.carmen.info.*;

/**
   This interface defines the callback methods consumers need to provide.
   When a request for a service provided by the consumer is received this
   function will be called along with the request attributes.
**/

public interface RequestHandler {

    /**
       The only callback function. Takes the service, query attributes, and
       an empty ServiceProvider.
       Returns true if it can service the request and fills in the
       ServiceProvider object. If not returns false
    **/
    public boolean handleRequest(String service, Vector attributes,
                                 ServiceProvider sp);
}

==================================================================================

package com.hp.carmen.api;
import java.util.*;
import com.hp.carmen.info.*;

/**
   This interface defines the callback methods consumers need to provide.
   When a request for a service provided by the consumer is received this
   function will be called along with the request attributes.
**/

public interface ResponseHandler {

    /**
       The only callback function. Takes the service, query attributes, and
       an empty ServiceProvider.
       Returns true if it can service the request and fills in the
       ServiceProvider object. If not returns false
    **/

    public boolean gotResponse(int id, String service,
                               ServiceProvider sp, Vector attributes);

}
```