Auto-Discovery Capabilities for Service Management: An ISP Case Study

Srinivas Ramanathan, Deborah Caswell, Scott Neal
Internet Systems and Applications Laboratory
HP Laboratories Palo Alto
HPL-1999-68
May, 1999

E-mail: [srinivas,caswell]@hpl.hp.com
scott_neal@hp-fortcollins-om4.om.hp.com

service management, auto-discovery, service models, Internet services

Auto-discovery is one of the key technologies that enables management systems to be quickly customized to the environments that they are intended to manage. As Internet services have grown in complexity in recent years, it is no longer sufficient to monitor and manage these services in isolation. Instead, it is critical that management systems discover dependencies that exist among Internet services, and use this knowledge for correlation of measurement results, so as to determine the root-causes of problems. While most existing management systems have focussed on discovery of host, servers, and network elements in isolation, in this paper, we describe auto-discovery techniques that discover relationships among services. Since new Internet services and service elements are being deployed at a rapid pace, it is essential that the discovery methodologies be implemented in an extensible manner, so that new discovery capabilities can be incrementally added to the management system. In this paper, we present an extensible architecture for auto-discovery and describe a prototype implementation of this architecture and associated auto-discovery techniques. We also highlight experiences from applying these techniques to discover real-world ISP systems. Although described in the context of ISP systems, the concepts described in this paper are applicable for the discovery of services and inter-service relationships in enterprise systems as well.

1 Introduction

The emergence of a variety of new services such as World Wide Web access, electronic commerce, tele-commuting, and virtual private networks has contributed to the phenomenal increase in Internet usage in recent years. In the process, the complexity of services offered by Internet service providers (ISPs) to their subscribers has increased dramatically. From supporting simple client-server application services (e.g., file transfers) that involve a single server application communicating over a network link to a client, ISPs are having to deal with services that involve multiple, complex relationships not only among their service components (application servers, hosts, network links, etc.), but also with other services. A classic example is the pervasive web service. Although to a subscriber, the web service appears to be provided by a web application server alone, there are a number of other services and service elements that contribute to this service. For instance, to access a web server, the subscriber first uses the domain name service (DNS) to obtain the IP address of the web site. The subscriber request is then forwarded to and handled by a web application server. The web page(s) being accessed may be stored on a back-end network file system (NFS), from where it is retrieved by the web server on demand. When the subscriber perceives a degradation in the quality of the web service, the problem may be due to any of the web service components (the web server host, the web application server, the network links interconnecting the subscriber to the web server host), or due to the other infrastructure services on which the web service depends (e.g., DNS and NFS).

Unfortunately, the majority of management systems have not kept pace with the evolution of services [1]. Most existing management systems lack the capability to capture and exploit the inter-relationships that exist among Internet services, and hence, offer very little support for monitoring and diagnosis of Internet services [2].

1.1 Utility of Service Models

One of the ways of representing the dependencies among services and service components in an ISP system is using a service model that captures the logical topology of a service [3]. Figure 1 depicts a service model for the web service offered by an ISP from a host named *www.isp.net*. In this model, nodes represent the health of different services and service elements that impact the web service. Arrows inter-connecting nodes represent dependencies among the nodes, while the arrows directly feeding a node represent measurements of the node. The root node (node N_1 in Figure 1) represents the overall health of the web service. Measurements of availability and performance assess the service health. The next level of the service model depicts the dependencies of the web service on the DNS and NFS services, represented by nodes N_2 and N_3 respectively in Figure 1. Since the state of the web application server too affects the web service, the service model depicts a dependency between the web service node (N_1) and the web application server node (N_4). In turn, the web application server's health depends on the state of the host machine (N_5) on which it is executing – over-utilization of the host could result in a slow-down of the web application server, thereby affecting the web service itself.

The hierarchical structure of a service model greatly simplifies the task of root-cause analysis: by correlating the health of services and service components based on their location in the service model, ISP operators can diagnose problems quickly. The utility of a service model stems from its ability to represent relationships that exist between services and service components. One of the key challenges inhibiting the pervasive applicability of service models is the difficulty in discovering the relationships that exist among services in an ISP system and constructing custom models for each

environment that is to be managed. Handcrafting a service model requires a great deal of effort and is time-consuming. Often, ISP personnel do not have the time or the ability to craft elaborate service models to monitor their systems. Therefore, a pre-requisite for any management solution targeted at ISPs is its ability to be automatically customizable to the environment being managed, with minimal human intervention.

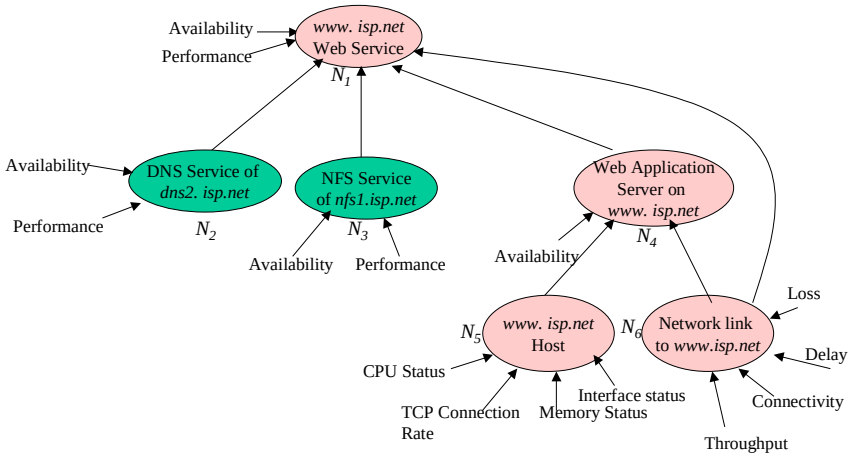

Figure 1: A service model for the web service offered by *www.isp.net*

1.2 Role of Auto-Discovery

Auto-discovery is one of the key methodologies that can be used to reduce the amount of human intervention required to customize a management system to a new environment. Pioneering work in this area is credited to Wu [4]. This work proposed a novel way of discovering individual hosts in a network by using SNMP queries directed at the network routers. Recent work by Siamwalla et al. proposes various heuristics for discovery of network topologies of Internet service providers [5]. While both the above efforts focus on discovery of network elements alone, many of the newer service management products, such as Hewlett-Packard's *Firehunter* [1], IpSwitch's *WhatsUp* [6], and Microsoft's *Internet Service Manager* include mechanisms to discover the existence of application servers on different hosts. By running active tests that connect to a number of commonly used transport protocol (the User Datagram Protocol – UDP, the Transmission Control Protocol - TCP) ports, these products discover application servers. Another way of discovering the existence of application servers is using special-purpose management agents that may be pre-installed on the different hosts in the network. *Unicenter TNG* includes agents that passively monitor processes executing on the servers and report back to a central management console. Very recently, the Internet Engineering Task Force has begun the definition of a new Service Location Protocol that application developers can adopt to enable discovery of their applications [7]. The focus of all the above efforts is predominantly on the discovery of application services *in isolation*.

1.3 Contributions of This Work

This paper focuses on methodologies for auto-discovery of inter-dependencies among services in an Internet system. Section 2 introduces the target environment being considered in this paper, while Section 3 discusses the inter-dependencies that exist in an Internet system. For discovery of inter-dependencies, in Section 4, we propose a novel two-phase discovery approach in which the first phase takes an external view of the ISP system and discovers the different

services that exist in the system. Some of the service components and service inter-dependencies are also discovered in this phase. Since many service inter-dependencies cannot be discovered by an external view, a second phase of discovery that uses software agents executing within the ISP system complements the first phase. Specific discovery techniques that can be employed in the two discovery phases are discussed in Section 5.

Most existing management systems implement discovery capabilities in a monolithic manner, making it difficult to extend the auto-discovery capabilities of the base system to handle new services. In addition, since the discovery logic is not exposed, it is difficult to modify or enhance the discovery capabilities of these systems. Since new Internet services and service elements are being deployed at a rapid pace, it is essential that the discovery methodologies for an Internet service management system be implemented in an extensible manner. To achieve this goal, in Section 6, we describe a template-driven approach that enables new discovery capabilities to be added to the management system over time. Section 7 describes a prototype implementation used to validate the concepts described in this paper by targeting several real-world ISP systems. The results of the auto-discovery process can be used to automatically generate service models that are customized to the environment being managed. A detailed description of a methodology for automatically generating a customized service model based on auto-discovered information is outside the scope of this paper (see [3] for details).

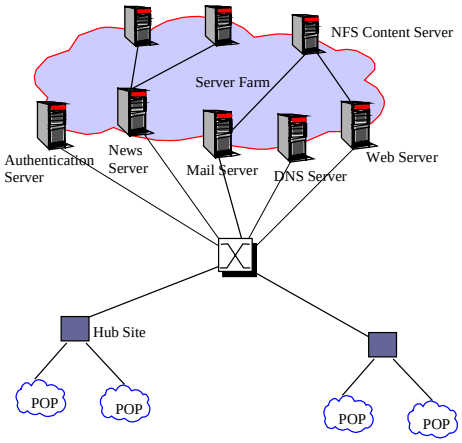

Figure 2: A typical ISP system

2 Target Environment

An ISP system has two main components (see Figure 2):

- ***Points of Presence (POP):*** In order to allow subscribers to connect to the Internet, an ISP establishes strategically located sites called Points of Presence (POPs) that house modem banks to which subscribers connect through the telephone network using dial-in modems. Each modem bank in the POP site is associated with a terminal server, which is a device that handles communication of packets from the POP to other locations in the ISP system. When a subscriber connects to one of the modem lines in the POP, the corresponding terminal server performs subscriber authentication and then assigns an IP address to the subscriber PC.
- ***ISP Server Farm:*** A server farm is a location housing servers that support applications such as Web, Email (Pop3 and SMTP), and News that residential subscribers access. Web sites that the ISP hosts for business customers are also

located in the server farm. Infrastructure services such as authentication during login, domain name service (DNS), content storage via the network file system (NFS) etc., are also offered via servers at this site.

3 Discovery Requirements

Several characteristics of the target environment influence the choice of the discovery mechanisms that must be employed. First, ISP systems are heterogeneous, employing a number of operating systems (Solaris, NT, FreeBSD, Linux, etc.), hardware platforms (Sun, HP, Intel), and software applications (e.g., Apache Web server, Netscape messaging server, etc.). Consequently, the discovery mechanisms included in a management solution for ISPs must be general enough to function in all such environments. Moreover, in order to be applicable pervasively, the discovery mechanisms should not require changes in existing applications and network equipment.

The choice of discovery mechanisms is also influenced by the nature of inter-dependencies that exist in an ISP system. The various dependencies that exist in an ISP system can be classified as:

- *Execution dependencies*: A discovery mechanism for ISP systems must discover which of the different types of application servers (Web, Email, News, DNS, NFS, Radius, etc.) are executing on each of the host machines.
- *Component dependencies*: In order to ensure scalability and redundancy, an ISP service (a parent node in a service model) may be comprised of more than one services of the same type (child nodes in the service model). Since the latter services are components of the former, dependencies of this type are component dependencies. A typical example of a component dependency occurs in the deployment of Web, Email, and News services. To ensure the scalability of their services, ISPs often replicate Web, Email, and News content across a number of servers. A common technique for balancing load among these servers uses the round-robin scheduling capabilities offered by the DNS service [8]. In this technique, replicated servers that support the same service (Web, Email, or News) are grouped together and assigned a single domain name in the DNS database. A DNS server that receives a request for this domain name returns the IP address of one of the servers in the group corresponding to the domain name in a round-robin manner.

  Since subscribers typically access services using domain names rather than IP addresses, the ISP's use of a group of replicated servers is often invisible to the subscribers. In this paper, we refer to the service provided by a group of servers functioning cooperatively as a *service group*. Depending on whether all of its component IP addresses support Web, Email, or News services, a domain name may be a Web service group, an Email service group, or a News service group, respectively. The discovery mechanism must be intelligent enough to discover such component dependencies.
- *Inter-service dependencies*: As explained earlier, a service may depend on other services for its proper operation. For example, the web service depends on a DNS service to allow the subscriber to connect to the web server host using its IP address and an NFS service to access the web content. Discovery of such inter-service dependencies is critical for effectively managing ISP systems.
- *Organizational dependencies*: Typically, ISP have different operations personnel who are responsible for the different services and service components. For example, an ISP may have an expert managing the Web service, another one managing DNS, another for NFS, and so on. Operational responsibilities may also be delegated based on

the geographic location of the service components. Responsibilities may also be partitioned among ISP staff depending on whether the ISP servers are accessible from the external global Internet, or whether they are accessible only from within the ISP system. The management system should be customizable to the ISP's organizational structure, so that operations personnel only view the states of the services and service components that they are responsible for. Furthermore, the customization must be achieved without much effort on the part of the ISP personnel at the time the management system is set-up. Since the precise organizational structure may vary from one ISP to another, the discovery mechanism must provide a means by which it can quickly learn the ISP's organizational structure.

4 Discovery Methodologies

Figure 3 depicts the discovery methodologies we propose. The first phase of discovery adopts a "black-box" approach to discovery. Predominantly this phase involves active tests that view the ISP system as a "black box" and attempt to discover the existence of different types of servers (i.e., the execution dependencies) by generating test traffic to all of the ISP hosts. Since the active tests do not require additional support from the ISP hosts and servers for discovery, this phase of discovery can be executed even from locations external to the ISP hosts. Hence, we use the term *external discovery* to refer to this first phase of discovery. As we shall see later, the external discovery phase can discover most of the component and organizational dependencies, as well as a few inter-service dependencies, in addition to the execution dependencies.

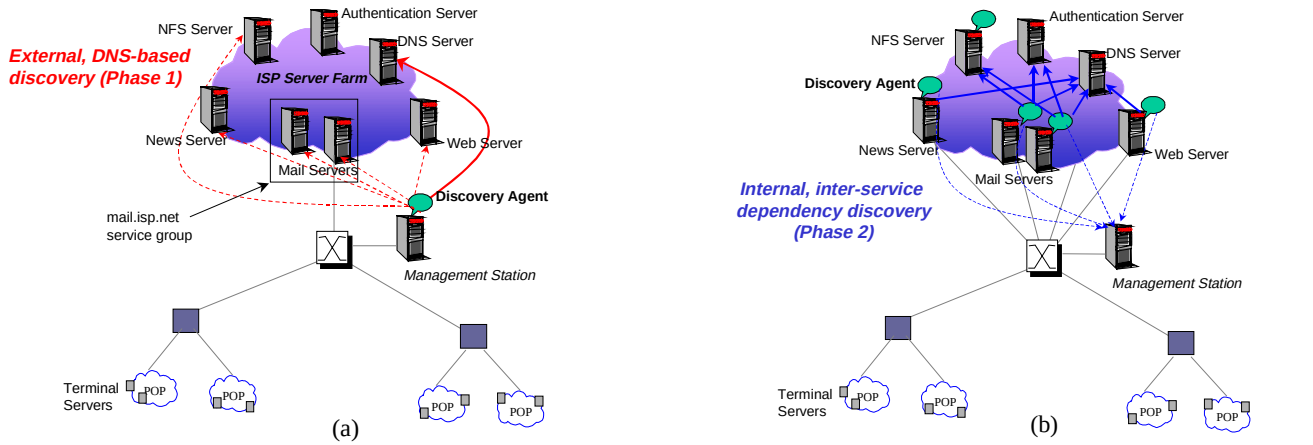

Figure 3: (a) External discovery: The solid line depicts a discovery agent contacting the ISP's DNS server to get a list of hosts in the ISP system, while the dashed lines indicate subsequent active tests being executed by the discovery agent to discover various types of dependencies. (b) Internal discovery: The solid lines indicate the discovery of inter-service dependencies using discovery agents installed on the ISP hosts. The dotted lines indicate the flow of discovered information back to the management station.

Since it may not be possible to discover all the inter-service dependencies that exist using tests executed from outside the ISP hosts, our architecture uses a second phase of discovery that complements the first phase. Taking an internal view of the ISP system, this phase discovers all the other inter-service dependencies that may exist. Since it takes an internal view of the ISP system, this phase is referred to as the *internal discovery* phase. The two phases of discovery are executed sequentially, with the second phase utilizing the discovered information output by the first phase to direct its operation.

Different mechanisms can be employed in the internal discovery phase. Figure 3 (b) depicts an example in which special-purpose discovery agents deployed on the ISP host machines yield information about dependencies that they can detect.

5 Discovery Techniques

In this section, we describe specific discovery techniques that can be employed in the two phases of discovery.

5.1 External Discovery

A component of an ISP system that is often neglected for the purpose of discovery is the domain name service. In order to allow subscribers to access hosts using their names, rather than by specifying the IP addresses, the DNS service is used. In addition to mapping host names to IP addresses, the DNS service also maintains information about a number of other relationships in the ISP system. For instance, the exchange of Email messages between hosts happens using the mail exchange records (MX records) maintained by the DNS. Furthermore, by their very definition, service groups are enabled via round-robin scheduling mechanisms implemented in the DNS servers. In ISP systems that host web sites for business customers, the ISP's DNS servers are responsible for advertising and supporting DNS queries for the customer web sites. In many cases, multiple web sites may be hosted on a single host, and the DNS service is responsible for mapping queries to the different web sites names to the common host. The DNS service also maintains information that identifies some of the name and mail servers of the ISP as external servers that are accessible from the global Internet. In summary, the domain name service holds a wealth of information that is critical for auto-discovery of ISP services.

A key innovation in our approach to external discovery is the use of the domain name service as a source of information about an ISP system. Since the DNS service does not maintain all the information necessary for external discovery (for instance, the DNS service cannot identify all the hosts that have web application servers executing on them), additional mechanisms are necessary to complement the DNS-based discovery mechanisms. The following sections describe the various discovery mechanisms in detail.

5.1.1 Host Discovery

One of the first steps in discovery is to determine all the hosts that exist in the ISP system. Most existing network management systems have taken one of two approaches:

- By scanning an address range (that is either specified by an operator or is determined based on the local subnet of the measurement host), and using the ICMP-Echo protocol, a management system can determine the hosts that are up.
- Alternatively, the default gateway of the management system can be used to bootstrap discovery. By querying the gateway's routing tables using SNMP, a management system can discover the hosts that exist in the local subnet. The routing information can be used to discover other routers, and the above procedure can be recursively applied to discover the entire network.

An approach that is complementary to the above approaches is to obtain a list of all hosts in the ISP system using information available with the domain name service. From the host name of the management station, the discovery process executing on the management system can deduce the domain name of the ISP system (e.g., the host *www.isp.net* exists in the domain *isp.net*). Using the default name service configured for the management system, the discovery process queries obtains the name server (NS) records that list the authoritative name servers for the ISP domain. These

name servers are authoritative in the sense that they have knowledge of all the hosts in the ISP system and are responsible for resolving queries that provide the IP addresses for these hosts into the corresponding domain names. By contacting one of these name servers, the discovery process obtains a list of hosts in the ISP system. Zone transfer capabilities supported by all DNS servers are used for this purpose [8].

5.1.2 Application Server Discovery

The existence of application servers of different types (Web, Email – Pop3 and SMTP, News, FTP, DNS, NFS, Radius, etc.), is verified by active tests that emulate typical client requests that are directed at the different TCP and UDP ports corresponding to the different service types. Information contained in the responses returned by the servers to the active tests can be used to discover additional information (such as vendor and version details) about the application servers executing on these hosts. The server responses have to be interpreted in a service-specific manner. For example, by observing the header in the response from a web server, the discovery process determines the type of web server (Apache, Netscape, Microsoft etc.) that is executing on the host machine. This information is used in two ways:

- *To customize the second phase of discovery:* For example, discovery agents installed on the ISP host machines in the internal discovery phase may process a web application server's configuration files to discover NFS dependencies that the server may have. Since web server configuration files are typically specific to each vendor, the server type information determined during external discovery can be used to determine the capabilities of the discovery agent(s) that must be deployed for internal discovery.
- *To customize the deployment of measurements for monitoring:* The server type information may also be used to determine specific measurements that must be targeted at the server to monitor its status. For example, Netscape web servers can be monitored by tracking variables specified in the Netscape-specific MIB that these servers support.

5.1.3 Service Group Discovery

Following the discovery of application servers, it is essential to determine which of these servers forms a part of a service group. By their very definition, service groups are configured using the DNS service. Hence, the existence of Web, Email, News, DNS, and other service groups has to be determined using DNS. By querying the DNS database, the discovery process determines a list of domain names that have multiple IP addresses associated with them. For each name in the list, the discovery process then determines whether each of its IP addresses hosts a common application server. If so, the name is most likely to represent a DNS round-robin service group that supports a common service. For example, suppose that all of the IP addresses corresponding to the name *www.isp.net* hosts a web server. In this case, *www.isp.net* represents a web service group. Note that in this process, a host that has multiple network interfaces, and hence, is assigned multiple different IP addresses may be improperly listed as a service group. The existence of such hosts can be discovered in the internal discovery phase, and these hosts can be removed from the list of service groups.

5.1.4 Mail Service Specific Discovery

The external discovery phase also discovers some dependencies that are specific to mail services in an ISP system. One of the critical measures of the performance of an ISP's Email service is the round-trip delay between the transmission of an Email message from a source host and its reception at the intended destination [1]. This measurement can be used to assess Email delivery times within the ISP domain, from the ISP domain to locations on the Internet, and from locations

on the Internet to the ISP domain. Since the Email service uses different application servers to send mail (SMTP-based servers) and to receive mail (servers based on the Pop3 or IMAP protocols), in order to initiate round-trip delay measurements of Email, it is essential to determine the relationships between the different types of Email servers in the ISP domain (i.e., which SMTP server can be used to send mail to a Pop3/IMAP based server?). Since mail forwarding is predominantly based on the MX records maintained in the DNS database, by querying the DNS service for MX records corresponding to each of the Pop3/IMAP servers, the discovery process determines the mail service relationships in the ISP domain.

The MX records in the DNS database are also useful in discovering organizational dependencies specific to the mail service. Typically, ISPs designate some of the SMTP mail servers as gateways that are responsible for the receipt of and the delivery of mail to sites outside the ISP system. These mail gateways are distinct from the servers that subscribers of the ISP access to receive and send mail, and may even be managed by different operations personnel. The mail gateways of the ISP system are discovered by querying the DNS servers for the MX records corresponding to the ISP's DNS system.

5.1.5 Terminal Server/POP Site Discovery

Various approaches can be adopted to discover the terminal servers in an ISP system. The most straightforward approach uses SNMP queries to obtain the MIB-II system description from all the hosts in the ISP network. Based on the replies it obtains, the discovery process can identify the hosts that are terminal servers. An alternative approach is based on the observation that because they need to operate and manage thousands of terminal servers, most ISPs having specific naming conventions that they use when naming their terminal servers. In fact, the naming convention more often indicates the association between terminal servers and POP sites, so that when a problem is reported by a subscriber using a POP site, the ISP operations staff can quickly decide which of the thousands of terminal servers they need to check to diagnose the problem. For example, one ISP[1] has chosen to use the convention *pm<number>.<POP site>.isp.net* to name their terminal servers, where the ISP's domain is *isp.net* (The terminal servers are Port Masters manufactured by Lucent Technologies – hence, the prefix *pm* in the name). With this approach, an ISP provides a regular expression representing the naming convention used as input to the discovery process. By matching the list of hosts that it has discovered with the naming convention, the discovery process not only determines the terminal servers that exist, but also determines the POP site to which a terminal server is assigned. Another key advantage of this approach is that it performs discovery without generating any additional network traffic.

5.1.6 Categorization of ISP Services

The approach adopted above, of exploiting naming conventions used by ISPs, can be extended to discover how ISPs categorize their other services as well. As an example, for each web site that it hosts for its business customers, an ISP has chosen to assign internal names of the form **.com.isp.net.* For instance, if this ISP hosts a web site named *www.customer-domain.com*, there is a corresponding entry for *www.customer-domain.com.isp.net* in the ISP's internal DNS database. As in the case of terminal servers, by permitting an ISP to specify their naming conventions, the discovery process composes a categorization of services that is custom to the target ISP system. This categorization can be based on

[1] We use the fictitious domain name *isp.net* to mask the identities of the ISPs referenced in the examples in this paper.

geographical locations of services (e.g., services offered by an ISP from a server farm in Dallas and those offered from a server farm in San Jose), based on business relationships (e.g., POP sites that *isp.net* outsources for a content provider), or based on the delegation of responsibilities among operators. The categorization information can be used to automatically define a customized service model for each ISP, with special nodes in the service model representing a collection of nodes pertaining to the same category.

5.2 Internal Discovery

The internal discovery phase focuses on discovering the inter-service dependencies that cannot be discovered by taking an external, "black-box" viewpoint. For instance, the relationship between a mail server and an NFS server is not discoverable from outside the ISP's system. There are two basic approaches for internal discovery:

1. *Using network probes:* Software probes installed at strategic locations on the network can snoop on packet transmissions. Since most TCP/IP communication is based on source/destination port numbers, by processing the headers of packets it captures, a software probe can deduce many of the relationships that exist among services. For example, a UDP packet transmitted from a mail server to the NFS port of an NFS server indicates that the mail server depends on the NFS server for its content storage. The main advantages of this approach are:

   - This approach enables discovery of inter-service dependencies independent of the specific types of application servers residing on the hosts.
   - Since it relies on just the ability to capture packets on the wire, this approach handles UDP and TCP-based services equally well.

   The main drawbacks of this approach are:

   - The use of switched network technologies by ISPs for interconnecting their servers constrains the utility of this approach. To capture all communications in a switched environment, probes may need to be installed on each of the segments of the switched network, making this approach very expensive.
   - To minimize costs, network probes can be installed on the ISP hosts themselves. However, there are some interesting challenges in designing probes in such a way that they do not adversely impact the performance of the services they are intended to discover.
   - IP security features that are likely to become prevalent in the near future are expected to make it impossible for network probes to decipher the source/destination port numbers in IP packets.
   - Discovery of inter-service relationships that may not be based on port numbers, e.g., the dependencies among virtual web servers operating on the same host machine, may not be discoverable by just processing the source/destination port numbers on IP packets.

2. *Using special-purpose discovery agents:* This approach relies on special-purpose agents installed on the ISP hosts for discovering relationships among services. The key difference between discovery agents and network probes is that unlike the probes, the discovery agents do not snoop on packet transmissions. Instead, the discovery agents use a number of operating system and application-specific mechanisms to discover inter-service dependencies. These mechanisms include:

- *Processing service configuration information:* Application servers figure out their dependencies on other services from one or more configuration files. By processing the content of the configuration files, discovery agents can discover inter-service dependencies. An example of this approach is the processing of a web server's configuration file to discover whether it has a dependency on an NFS service. While processing the web server's configuration file, a discovery agent can also determine if the same application is being used to host multiple "virtual" web sites. Since application configuration files are vendor-specific, vendor, version, and other application-specific information discovered during external discovery is used for deciding the location and format of the configuration files.

  The web server example highlights a *forward-looking* discovery agent - the discovery agent is forward looking in the sense that it determines dependencies of a service (web service in this example) on other services (e.g., NFS service) by querying configuration files of the application *providing* the service (the web server application). Sometimes it is easier to implement *backward-looking* discovery agents. These agents discover the dependencies on a service by querying the configuration files of applications that are *using* the service. For example, the configuration file of a mail authentication server may indicate which of the mail servers are using the authentication server.

- *Utilizing application-independent monitoring tools:* Rather than processing configuration information in an application-specific manner, sometimes it is easier to use monitoring tools available on the ISP server to discover inter-service dependencies. This approach is especially attractive for services that communicate using TCP. The popular *netstat* utility can be used to determine the TCP connections that exist on an ISP host. A discovery agent that executes this tool periodically can discover information about the source and destination ports and the host locations for TCP connections, which can then be used to deduce inter-service dependencies. This approach exploits the fact that most TCP implementations enforce a 3min delay for connections in the TIME_WAIT state of TCP [9], so that a connection persists for about 3 mins even after it is no longer in use. Consequently, whenever the discovery agent is executed, it is likely to detect all the TCP connections that may have been established in the 3 mins prior to its execution. This same approach does not work for UDP-based services, since UDP is connectionless, and there is no state that is maintained at either the source or the destination. This approach for monitoring TCP connections can be used to discover dependencies such as those that exist between mail servers and mail authentication servers, between web servers and back-end databases, between Radius/TACACS authentication servers and terminal servers, etc. As in the previous case, discovery agents can be either forward-looking or backward-looking.

The advantages of discovery agents compared to probes are:

- Since they do not require processing of individual packets, they impose lower overheads on the ISP hosts.
- There are fewer security concerns when using discovery agents. Probes require promiscuous access to the network traffic and may need to be installed with super-user privileges enabled.
- Unlike probes, discovery agents will continue to function even in environments in which IP security is enabled.
- The overhead and cost of deploying discovery agents is independent of the interconnecting network technology.

- Finally, all of the discovery agents do not need to be deployed at the same time. Instead, the deployment of discovery agents can happen at the discretion of the ISP. As and when new discovery agents are installed on the ISP hosts, additional information is discovered about the ISP system.

The main drawback of discovery agents is that application-specific discovery agents may be harder to develop, given the plethora of applications that are available in the market. The choice of approaches for internal discovery may depend on the ISP system being discovered – the network topology in use, security restrictions in effect, types of application servers in use, etc. Owing to the simplicity of their implementation, our prototype implementation (Section 7) uses discovery agents.

6 Extensible Discovery Architecture

As indicated earlier, since new Internet services and service elements are being deployed at a rapid pace, it is essential that the discovery methodologies be implemented in an extensible manner, so that new discovery capabilities can be incrementally added to the management system. Figure 4 depicts an extensible architecture for discovery. The three main elements of this architecture are:

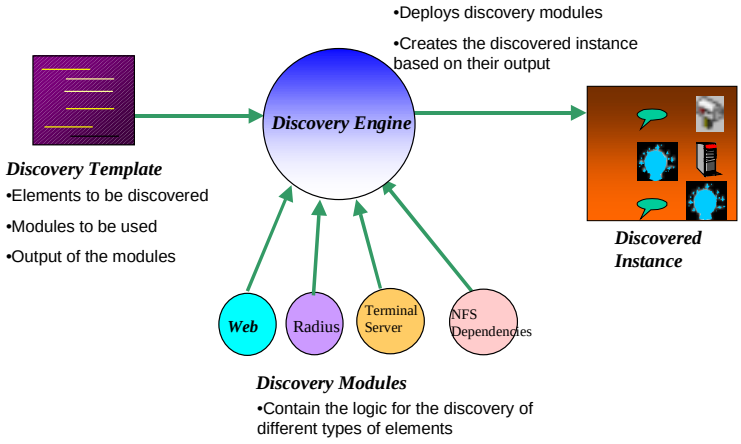

Figure 4: Auto-discovery architecture

- *Discovery modules*: The logic used for discovery of different services and service elements (i.e., the specific techniques described in Section 5) is encapsulated in the discovery modules.
- A *discovery template* is a key to the extensibility of the auto-discovery architecture in the sense that it drives how discovery is performed. The template defines the different services and service elements that need to be discovered and the specific discovery modules that can be used to discover these elements.
- A *discovery engine* drives the auto-discovery. The discovery engine interprets the discovery template and for each service or service element type specified in the discovery template, it invokes the corresponding discovery modules specified in the template. All of the discovery modules report the results of their execution back to the discovery engine. The discovery template contains instructions for the discovery engine to process the output of the discovery modules and record them as a *discovered instance*.

Some discovery modules may rely on the discovery results of other discovery modules – for example, a DNS round-robin service group discovery module for Web services relies on knowing which hosts support web services (an output of the Web service discovery module). The discovery engine facilitates communication among discovery modules, receiving discovery results from one module and forwarding the results to other modules that may be interested in the results. In contrast to the discovery modules, the discovery engine is designed to be independent of the services that need to be discovered. Consequently, to discover new services or service elements, a user has to provide a discovery template specification and one or more discovery modules for these new elements. By providing an alternate discovery module for a service that is already supported, a user can also enhance the capabilities of an existing discovery system.

7 Prototype Implementation

In this section, we describe our prototype implementation of the auto-discovery architecture described in the previous section. We present example specifications for a discovery template and a discovered instance, as well as a detailed design for the discovery engine. Both the discovery template and the discovered instance in our prototype are specified using the INI file format that is commonly used to represent the content of system files in Microsoft Windows-based personal computer systems. As per the INI format, a template or an instance is organized as different sections, each of which is specified by a unique string enclosed within a pair of square brackets ("[<*section name*>]"). Within each section, a number of variable-value pairs are specified. This syntax is only an example. Other specification and schema definition technologies such as Common Information Model (CIM) being proposed in the Desktop Management Task Force [10] can also be used to implement the concepts described in the following sections.

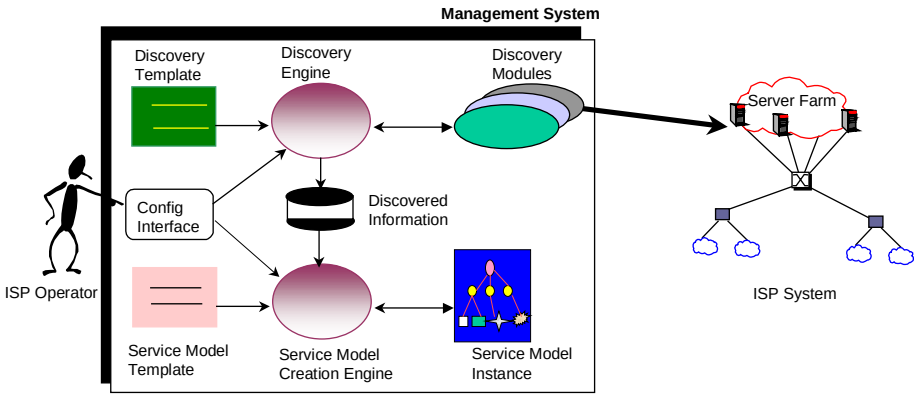

Figure 5: Architecture of the prototype system

7.1 Overview

Figure 5 presents an overview of our prototype system. Our prototype has two main components - a discovery engine and a service model creation engine. The discovery engine is responsible for generating a discovered instance by discovering the services and service components and their inter-dependencies in a target ISP system. Owing to the absence of direct access to a real-world ISP system (necessary for implementing internal discovery), our prototype discovery engine only implements external discovery. The service model creation engine is responsible for generating a service model instance

using the discovered instance output by the discovery engine. The service model creation engine is implemented in an extensible manner, following the architecture described in [3]. In this approach, a service model template outlines the service topology for a service. By mapping the discovered services and service inter-dependencies to different nodes in the service model template, the service model creation engine generates a service model instance that is customized to the ISP system being managed. Details of the service model construction methodology are discussed in [3].

7.2 Discovery Template Specification

Figure 6 depicts an example discovery template specification. Each section defines how discovery of a specific service or service element should happen. For example, sections in the specification in Figure 6 represent templates for discovery of external name servers, hosts, Web servers, Mail Pop3 servers, etc. The *module* variable specification in each section identifies the discovery module that the discovery engine must invoke to discover elements of the type specified in the section headings. The *arguments* variable represents arguments that are passed by the discovery engine to the discovery module during invocation. The *outputs* variable defines the number and names of the columns in the discovery module's output. The discovery modules are assumed to output tab separated column values that the discovery engine can parse and output according to a discovered instance output format (defined later).

Notice from Figure 6 that the outputs for each section can be very different. For each element that it discovers, a discovery module outputs a row of output values. The *instanceKey* variable represents the index that is associated with each row of output values. The column name(s) specified on the right-hand side of the *instanceKey* assignment must correspond to one of the column names that are specified in the *output* assignment.

```
[ExternalNameServers]
; Discovery of External name servers
discovery-module=DiscoverExternalNameServers.class
; the module used for discovery
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
; location of the Web interface from which the discovery module
; obtains configuration information
discovery-outputs=ipAddress, hostName, domainName, category
discovery-instanceKey=<ipAddress>:DNS
; the key used to refer to instances of this type
discovery-dependencies=
; this discovery module does not depend on other modules

[Hosts]
; discovery of Hosts in the ISP system
discovery-module=DiscoverHosts.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, state, realName, domainName, category
; ipAddess and hostName are strings that are used to refer to the host
; state indicates whether a host is alive or not
; realName is the unique name of the host, which is mainly used to discover virtual servers (not shown in this example)
discovery-instanceKey=<ipAddress>:Host
discovery-dependencies=ExternalNameServers
; this module depends on discovery of external name servers

[SMTPServers]
; discovery of SMTP mail servers
discovery-module=DiscoverSmtpServers.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, serverType, realName, category
; ipAddress and hostName refer to the host on which the mail server executes
; serverType refers to the mail server application (e.g., Netscape, Microsoft)
; realName is the name of the host as returned by the SMTP server (used to detect virtual servers)
; category is used to classify the mail server – can represent geography, also used
;      to identify the external mail servers
discovery-instanceKey=<ipAddress>:Smtp_Mail
discovery-dependencies=Hosts,ExternalNameServers

[POP3Servers]
; discovery of Pop3 mail servers
discovery-module=DiscoverPop3Servers.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, serverType, realName, relatedSmtpServer, category
; ipAddress and hostName refer to the host on which the mail server executes
; relatedSmtpServer indicates the SMTP server that must be used to send mail to this Pop3 server
discovery-instanceKey=<ipAddress>:Pop3_Mail
discovery-dependencies=Hosts,ExternalNameServers

[NewsServers]
; discovery of news servers
discovery-module=DiscoverNewsServers.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, serverType, realName, category
discovery-instanceKey=<ipAddress>:News
discovery-dependencies=Hosts
```

```
[WebServers]
; discovery of web servers
discovery-module=DiscoverWebServers.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, serverType, category
discovery-instanceKey=<ipAddress>:Web
discovery-dependencies=Hosts

[InternalNameServers]
; discovery of internal name servers
discovery-module=DiscoverInternalNameServers.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, domainName, category
; domainName refers to the domain of the ISP system
discovery-instanceKey=<ipAddress>:DNS
discovery-dependencies=Hosts

[TerminalServers]
; discovery of terminal servers in the ISP's POP sites
discovery-module=DiscoverTerminalServers.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=ipAddress, hostName, popSite
; ipAddress and hostName identify the terminal server
; popSite is the name of the POP site that the terminal server exists in
discovery-instanceKey=<ipAddress>:TerminalServer
discovery-dependencies=Hosts

[POPSites]
; discovery of POP sites
discovery-module=DiscoverPOPSites.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl
discovery-outputs=popSite
; this module processes the terminal server discovery module's output to discover the POP sites
discovery-instanceKey=<popSite>:POPSite
discovery-dependencies=Hosts,TerminalServers

[WebServiceGroups]
; discover Web service groups
discovery-module=DiscoverServiceGroups.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl -service Web
discovery-outputs=serviceGrpName, serviceGrpComponentsList, category
discovery-instanceKey=<serviceGrpName>:WebServiceGroup
discovery-dependencies=WebServers

[Pop3_MailServiceGroups]
; discover Pop3 mail service groups
discovery-module=DiscoverServiceGroups.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl -service Pop3_Mail
discovery-outputs=serviceGrpName, serviceGrpComponentsList, category
discovery-instanceKey=<serviceGrpName>:Pop3_MailServiceGroup
discovery-dependencies=POP3Servers

[Smtp_MailServiceGroups]
; discover SMTP mail service groups
discovery-module=DiscoverServiceGroups.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl -service Smtp_Mail
discovery-outputs=serviceGrpName, serviceGrpComponentsList, category
discovery-instanceKey=<serviceGrpName>:Smtp_MailServiceGroup
discovery-dependencies=SMTPServers

[NewsServiceGroups]
; discover News service groups
discovery-module=DiscoverServiceGroups.class
discovery-arguments=-url http://ism-pc2/scripts/discEngineAPI.pl -service News
discovery-outputs=serviceGrpName, serviceGrpComponentsList, category
discovery-instanceKey=<serviceGrpName>:NewsServiceGroup
discovery-dependencies=NewsServers
```

Figure 6: An example discovery template specification

The *dependencies* variable in a template indicates the dependencies that a discovery module has on other modules. For instance, the Web server discovery module has a dependency on the host discovery module. As we will see later, this information can be used by the discovery engine to decide the sequence in which discovery modules must be scheduled.

7.3 Discovery Engine Design

There are two contrasting approaches for designing the discovery engine and discovery modules.

- *Discovery engine driven discovery:* In this model, the discovery engine controls the discovery process. The discovery engine executes periodically and every time it starts, the discovery engine processes the discovery template. By considering the values of the dependencies variable for each of the sections in the discovery template, the discovery engine determines the order in which the sections must be processed. Sections with no dependencies are processed first and their corresponding discovery modules are invoked. Then, the discovery engine iterates through the list of sections, choosing the sections that have not been processed by it and for which all of their dependencies have been

processed earlier for processing. This process is repeated periodically to discover new instances as and when they show up. Figure 7 depicts the logical building blocks of a discovery engine.

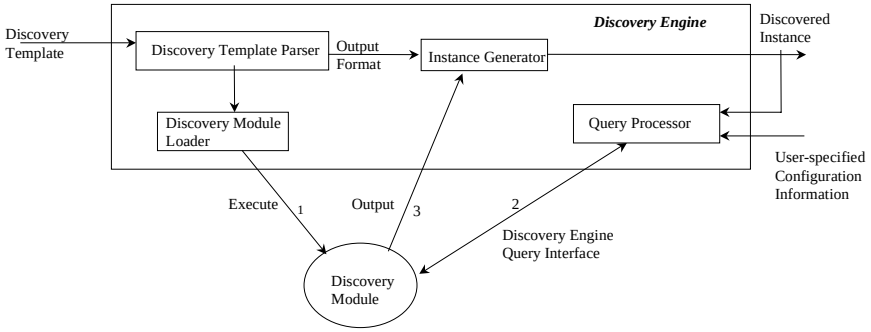

Figure 7: Building blocks of a discovery engine

In our prototype implementation, the discovery engine uses the exec system call to invoke the discovery modules as separate processes. By doing so, the discovery engine is able to handle discovery modules written in a variety of programming environments ranging from Java classes to perl scripts to shell scripts. The query processor component of the discovery engine performs two functions. First, when a module starts up, it queries the discovery engine to obtain user-specified configuration information that guides the operation of the discovery modules. Figure 8 depicts a typical configuration specification. The first column of a line in this specification identifies the discovery module to which the instruction on this line pertains. There are three types of instructions in the configuration specification. All of these instructions specify regular expression patterns that must be applied against the IP address or host name of a service or service element. The instructions in the configuration file are criteria that instruct the discovery modules to include or exclude specific services or elements, or criteria that indicate the naming convention used by the ISP.

```
Hosts           exclude     ip          10.1.204.1-10.1.205.1
# exclude all hosts with IP addresses in this range.  These hosts represent subscriber home PCs
WebServers      category    name        .*.com.fakeisp.net          WebHostingServers
# servers with names of the form *.com.fakeisp.net are Web hosting servers
TerminalServers extract     name        max[0-9]{3}<PopSite>[0-9]t.fakeisp.net
# Terminal servers must match the naming pattern above. Extract the POP site name from the terminal server's name
```

Figure 8: An example of a configuration file that guides the operation of the discovery modules

A second function of the query processor is to provide the discovery modules with access to previously discovered instances. Based on configuration and discovered instance information obtained from the query processor, the discovery modules perform tests on the ISP system and report their discovery output back to the discovery engine. The instance generator module processes the output from the discovery modules and creates the discovery instance in an appropriate format. The format of the discovery template and the discovery instance are thereby hidden from the discovery modules.

- *Discovery module driven discovery*: In this alternative model, the discovery engine processes the template once, invoking all the discovery modules simultaneously. From this point, the discovery modules determine when different elements in the ISP system are discovered. The discovery modules execute periodically, looking for new instances. Some discovery modules are independent in the sense that they are not reliant on other modules for discovery. These modules can begin executing immediately. As and when they discover new instances, the discovery modules forward

their results to the discovery engine. Based on the dependencies on a discovery module (specified in the discovery template), the discovery engine forwards the results obtained from a discovery module to other discovery modules that are interested in the results. The availability of new results (e.g., the discovery of a new host) may trigger discovery by other modules (e.g., the web server module checks to see if a web server is executing on the new host), and this process continues. A key advantage of this approach is that multiple discovery modules may be executing in parallel, discovering the ISPs services. In this approach, the discovery engine mainly functions as a facilitator of communication among the discovery modules.

7.4 Discovered Instance Specification

Figure 9 depicts an example of a discovered instance. Section names in the discovered instance correspond to the *instanceKey* variable specifications in the discovery template. For each of the *outputs* variables in the discovery template, there is an assignment in the discovered instance.

```
; External Name Servers
[10.174.173.23:DNS]
ipAddress=10.174.173.23
hostName=dns4.isp.net
domainName=isp.net
; the ISP domain name
category=ExternalDnsServer
; category indicates that the server is an external DNS server

; Hosts in the ISP system
[10.174.173.23:Host]
ipAddress=10.174.173.23
hostName=dns4.isp.net
state=Alive
; state of the host – Alive or NotAlive
realName=dns4.isp.net
; real name of the host
domainName=isp.net
… Similar definitions for [10.137.196.52:Host],[10.137.196.54:Host], [10.137.196.56:Host], and
[10.137.196.58:Web]

; SMTP mail servers in the ISP system
[10.137.196.52:Smtp_Mail]
ipAddress=10.137.196.52
hostName=mailfes21.isp.net:smtp.isp.net
serverType=
realName=mailfes21.isp.net
category=ExternalSmtpServer
; the category above identifies that this server is an external server (i.e.,
; a mail gateway)

; Pop3 mail servers in the ISP system
[10.137.196.52:Pop3_Mail]
ipAddress=10.137.196.52
hostName=mailfes21.isp.net:smtp.isp.net
serverType=qpopper 2.1
; type of Pop3 server
realName=mailfes21.isp.net
relatedSmtpServer=smtp.isp.net
; the SMTP server used to send mail to the Pop3 server
[10.137.196.54:Pop3_Mail]
ipAddress=10.137.196.58
hostName=mailfes22.isp.net
serverType=Netscape Messaging Server
realName=mailfes22.isp.net
relatedSmtpServer=smtp.isp.net
category=
[10.137.196.56:Pop3_Mail]
ipAddress=10.137.196.56
hostName=mailfes23.isp.net
serverType=qpopper 2.1
realName=mailfes23.isp.net
relatedSmtpServer=smtp.isp.net
category=

; Web servers in the ISP system
[10.137.196.58:Web]
ipAddress=10.137.196.58
hostName=www.isp.net
serverType=NCSA/1.5
; type of web server

; Terminal servers in the ISP system
[10.137.197.1:TerminalServer]
ipAddress=10.137.197.1
hostName=max001palo-alto1t.isp.net
popSite=palo-alto
```

```
; POP site where this terminal server is located

; POP sites in the ISP system
[palo-alto:POPSite]
popSite=palo-alto

; Pop3 mail service groups in the ISP system
[pop3.isp.net:Pop3_MailServiceGroup]
serviceGrpName=pop3.isp.net
serviceGrpComponentsList=10.137.196.52:10.137.196.54:10.137.196.56
```

Figure 9: An example of a discovered instance for an ISP

8 Integrating Discovery with Service Models

There are two ways in which discovery can be integrated with service models. A looser integration involves using the output of discovery, together with a service model template that outlines the structure of a service, to automatically generate a service model instance that is customized for the ISP system being managed. A detailed discussion of an approach for automatically generating custom service model is discussed in [3]. A tighter integration involves driving auto-discovery and service model instantiation from a common template. In this approach, for each node in the service model, corresponding discovery template specifications are provided. The discovery and service model specific components of the template can be processed by a single application. This approach towards tighter integration of discovery and service model templates is attractive for several reasons:

- The two-phase discovery methodology fits very well with the service model concept. The service model template can serve to constrain the discovery process in the first phase - only services specified in the service model template need to be discovered. The dependencies among services and service elements specified in the service model template can then be used to determine the specific dependencies that the discovery process needs to target to discover in the second phase. For instance, suppose the service model template for a web service indicates that the web service relies on DNS and NFS services. If the first phase of discovery determines that a host, say *www.isp.net*, has a web server executing on it, then based on the service model template, the discovery process can determine that in the second phase of discovery, it must target to discover the NFS and DNS services on which the web service provided by *www.isp.net* depends.
- The service model template can use many of the outputs of the discovery process (such as the type of web server executing on a host, the terminal server to POP site dependencies, etc.) [3]. Using a common template permits tighter syntax checking across the discovery and service model components of the template.
- Finally, the discovery sequence itself could be decided based on the service model template specification. The discovery process may perform a post-order traversal of the service model template from a root node down, discovering instances of node types lower in the service model template graph before moving on to the higher levels.

Exploring the issues involved in implementing a tightly integrated approach to discovery and service modeling is a topic for future research.

9 Conclusions

In this paper, we have described techniques for automatically discovering various types of dependencies that exist in an Internet Service Provider system. Discovery of these dependencies is the basis for the automatic construction of service models that are customized to the system and services being managed. We have described an extensible architecture for

discovery, to which new capabilities can be added as and when new services or service elements are added to an ISP system. Exploring the applications of the discovery architecture and techniques described in this paper to emerging ISP services such as web hosting, E-Commerce, virtual private networks, and voice-over-IP services, as well as to enterprise-specific applications and services is a subject of future research.

References

[1] S. Ramanathan and C. Darst. Measurement and management of Internet services using HP Firehunter. *In Proceedings of the IFIP Integrated Management Conference*, May 1999.

[2] R. Wetzel. Customers rate ISP services. PC Week. November 1997.

[3] D. Caswell and S. Ramanathan. Using service models for management of Internet services. *Hewlett-Packard Laboratories Technical Report - HPL-99-*. February 1999.

[4] J.C. Wu. Automatic discovery of network elements. U.S. Patent #519187. Hewlett-Packard Company. May 1990.

[5] R. Siamwalla, R. Sharma, and S. Keshav. Discovering Internet Topology. Cornell University Technical Report, July 1998.

[6] M. Avery. WhatsUp Gold enhances network supervision tasks. *InfoWorld*, Vol. 20, No. 4, Jan 26, 1998.

[7] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. Internet Draft. July 1998

[8] P. Albitz and C. Liu. *DNS and BIND*. O'Reilly Publications. 1992.

[9] W. Stevens. *TCP/IP Illustrated*, Vol. 1, Addison-Wesley, MA. 1994.

[10] The DMTF Common Information Model (CIM) subcommittee, http://www.dmtf.org/work/cim.html