Towards Generic Application Auto-discovery

Vijay Machiraju, Mohamed Dekhil, Klaus Wurster,
Jerremy Holland, Martin Griss, Pankaj Garg
Software Technology Laboratory
HP Laboratories Palo Alto

HPL-1999-80
July, 1999

E-mail: {vijaym,dekhil}@hpl.hp.com

application management, auto-discovery, modeling, model-based discovery, software agents

The increasing complexity of enterprise applications, the expanding number of networked machines, and the rapid deployment of Internet-based business applications (e-commerce), emphasize the importance and value of application management. One of the main problems in current application management products is the amount of time and effort needed to install and customize them. Application auto-discovery is a key technology for solving this problem. In this report, we present a generic approach to application auto-discovery along with some examples. Our approach is to create a model-based discovery engine that is driven by an application template model. While the application template model captures the variation from one application to another, the auto-discovery engine uses sophisticated mechanisms such as scoping to execute an invariant auto-discovery process.

Towards Generic Application Auto-discovery

V. Machiraju, M. Dekhil, K. Wurster, P. Garg, M. Griss, and J. Holland
Hewlett-Packard Laboratories
1501 Page Mill Rd MS 1U-14
Palo Alto, CA 94304
USA
{vijaym,dekhil}@hpl.hp.com

Abstract

The increasing complexity of enterprise applications, the expanding number of networked machines, and the rapid deployment of Internet-based business applications (e-commerce), emphasize the importance and value of application management. One of the main problems in current application management products is the amount of time and effort needed to install and customize them. Application auto-discovery is a key technology for solving this problem. In this report, we present a generic approach to application auto-discovery along with some examples. Our approach is to create a model-based discovery engine that is driven by an application template model. While the application template model captures the variation from one application to another, the auto-discovery engine uses sophisticated mechanisms such as scoping to execute an invariant auto-discovery process.

Keywords

Application management, auto-discovery, modeling, model-based discovery, software agents.

1. Introduction

Many enterprises use complex applications for their day-to-day operations. Almost all businesses use Email and Internet *access* as a means of communicating within and outside their enterprise. Internet service providers (ISPs) run email and web servers to provide email and Internet services to their customers. Payroll and other human resource systems are used to track employees and their records. Database systems are used to store enterprise data. Financial applications are used to keep track of revenues and profits of companies. Many businesses allow some of their transactions to be conducted online. For some businesses, online transactions or electronic commerce is the main source of revenue. All of these applications are distributed in nature, variable in size, customizable, and dynamic.

While all these aspects make modern distributed applications more powerful, they make them complex to manage at the same time. **Application management** is defined by Sturn and Winston [13] as "the process of directing, administering, or controlling the use of application systems, features, and elements." Application management consists of two main activities: **monitoring** the behavior of the system, and **controlling** that behavior by executing certain actions on the managed system.

Today, there are several commercially available application management systems. However, existing products lag behind the expectations of the industry. According to a survey [5], nearly 80% of all the management solutions are not fully implemented or customized to the requirements after 2 years of effort. It takes a large amount of resource to install, customize, and deploy a management system before it can be used to obtain useful metrics about application behavior.

Application auto-discovery is a technology that enables the creation of out-of-the-box management solutions. Existing approaches to application discovery fall into two classifications: (1) manual techniques and (2) application-specific auto-discovery techniques. While manual techniques have the limitation of being extremely time-consuming and laborious, application-specific auto-discovery techniques are not reusable across the discovery of multiple types of applications. In this paper, we introduce **generic application auto-discovery**, whereby a single generic auto-discovery engine can be used to discover multiple application types.

The rest of the paper is organized as follows: Section 2 defines application auto-discovery and provides more insight into existing manual and idiosyncratic auto-discovery techniques along with their drawbacks. Section 3 introduces generic application auto-discovery. In order to create a generic solution we adopt a model-driven approach by defining application variants in an application template model. In section 4, we describe the application template model in greater detail. Section 5 briefly describes our implementation of the model-driven auto-discovery engine and some of our experiments in discovering e-commerce and SAP applications. We summarize our conclusions in section 6.

2. Application Auto-discovery

Understanding the application to be managed is central to the whole issue of simplifying the installation, customization, and deployment of management systems. If management systems could automate the process of capturing the necessary information about an application as a first step before using that information to deploy management policies, we have accomplished the goal of creating ready-to-use management solutions.

Application auto-discovery is the process of automatically capturing the necessary information about the application and representing that information in a format that can be used by other management solutions. Capturing application information *automatically* means that there should be a systematic algorithm, which can be executed to discover pieces of information about the application and to put all the pieces together into a consistent representation of the entire application.

2.1 Manual Discovery

Discovery has always been a part of network, system, and application management. Most of the application-level discovery, however, is performed as a manual process and is often ignored as a distinct knowledge-gathering step that could be automated. Required information about specific aspects of the application is either formally listed and stored at the time the application is installed or created, or is informally grasped only in the minds of the installers. Some times, it is determined on a case-by-case basis by following some idiosyncratic techniques or by executing custom scripts.

Manual approaches to discovering applications have several disadvantages, particularly in the context of application management:

- *Time consuming*: Discovering all components of an application, determining their configuration parameters, and understanding the relationships between these components is a time consuming and laborious process.
- *Distributed intelligence*: The knowledge and expertise about enterprise applications is often distributed in the minds of several individuals in the enterprise.
- *Inconsistency*: Information about components that participate in multiple applications should be consistent with respect to those applications.
- *Lack of reuse*: Time is spent in repeating similar or identical manual processes over and over.

2.2 Application-specific Auto-discovery

There has been extensive work in auto-discovery targeted at networks and physical systems on a network. Network Node Manager from HP OpenView [14] has specialized in automatically discovering network elements and topology. Recent efforts have extended these techniques into non-SNMP based networks [12], ATM networks [8], and mobile networks [10]. Surprisingly, there has been very little work on auto-discovering application components, their roles and relationships. There has been some research on discovering behavioral aspects of applications using event-data analysis [3] and data-mining techniques [7]. A template-driven approach for discovering certain application components was developed earlier at HP Labs [11]. This approach was based on automating a few discovery mechanisms (e.g., DNS) targeting components of an ISP farm. The discovery mechanism in each of these cases is only applicable to addressing the problems associated with the discovery of certain type of applications. Having multiple auto-discovery mechanisms for different applications has disadvantages in some cases:

- *Multi-application management tools*: General-purpose management tools such as HP OpenView cannot take full advantage of these point products since the discovery mechanism has to be reinvented for every class of applications.
- *Reuse across applications*: The problem of reuse explained earlier with the manual approaches still exists largely in idiosyncratic auto-discovery approaches. The representations and methods in each of these approaches may be reused within the approach but not across the discovery of multiple applications.

3. Generic Auto-discovery

There are two essential pieces of information that are required as input to auto-discovery. The first is a description of “what to discover.” The second is “how to discover.” The auto-discovery mechanisms discussed so far have pre-canned discovery algorithms that are tuned to the discovery of particular applications. In other words, the information about what and how to discover are embedded into the discovery algorithms. In order to create generic auto-discovery that can be used to discover instances of any application, we should factor both the "what-to-discover" and "how-to-discover" aspects (**variants** that change from one application to another) and use that as the input to a generic auto-discovery engine. The generic auto-discovery engine implements the **invariants** that are common to the discovery of all applications. Examples of these invariants are interpreting the variants, scoping where the discovery process should be executed, executing the discovery process, and storing the discovered instances in a standard format. The engine should be robust enough to handle complexities of dynamic distributed environments that enterprise applications typically run into.

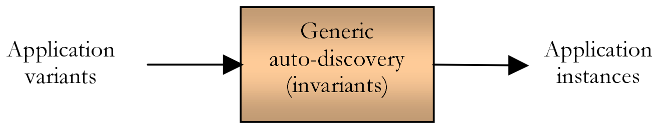


Figure 1: Generic auto-discovery

The variants should be represented in a standard format that can be understood by the auto-discovery engine. It should be easy to create, edit, or modify these variants as applications evolve. Object-oriented modeling offers a standard technique for representing application variants.

Modeling is a proven and well-accepted engineering technique to better understand a complex system [1]. Models provide a set of ground rules to work with while specifying such complex systems. They help in simplifying reality by offering structured mechanisms to represent the structure and behavior of these applications. By making the entities and relationships in applications *explicit*, they drive the generic auto-discovery process. They also help us visualize the systems, or enterprise applications in our case, by breaking them into comprehensible pieces that can be visualized at different levels of abstraction. Modeling also provides a unified representation of the application characteristics that can be used by different management systems, thus allowing reusability and generality of solutions.

One of the main factors promoting model-based solutions is the adoption of UML (Unified Modeling Language) by the industry and by standards organizations. More information about UML and its development history can be found in [1]. The Desktop Management Task Force organization (DMTF) introduced a standard information model for describing management of computer systems, using UML concepts, called CIM (Common Information Model) [9]. CIM is defined by DMTF as "a conceptual information model for describing management that is not bound to a particular implementation. This allows for the interchange of management information between management systems and applications." They also developed syntax for defining the managed objects called Managed Object Format (MOF). Following this, Microsoft provided an implementation of CIM on Windows platforms called WMI (Windows Management Initiative) [9]. They developed a CIM Object Manager (CIMOM) which can be viewed as a repository of management information. The introduction of CIM and WBEM opened the door for building generic, model-based management solutions, with the ability to interchange information about the managed objects through a standard representation and format.

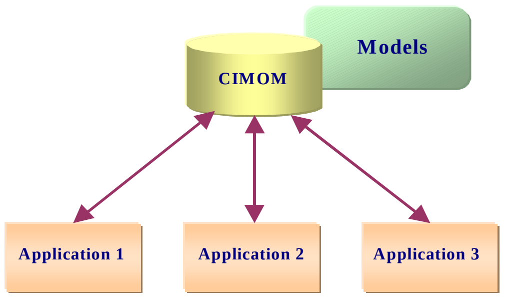


Figure 2: CIMOM acts as a repository for multiple application models

Our approach to generalized auto-discovery is to use models to represent application variants and the discovered application instances (Figure 3). The former model will be called **application template model** and the latter will be termed **application instance model**.

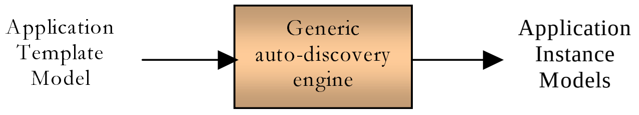

Application Template Model
Generic auto-discovery engine
Application Instance Models

Figure 3: Model-based auto-discovery

4. Application Template Models

Application template models provide a standard format for representing application specific information. This includes information about the structure of the application and about the discovery techniques that should be used to discover instances of the application.

4.1 Modeling Application Structure

The model for representing application templates consists of an **application template class** and multiple **application template objects**. The application template class defines the standard format that should be followed by every application template. Each application instantiates the application template class to define specific application template objects. Application template class and application template objects bear the same relationship as a class to its instances in traditional object-oriented programming systems. The UML representation for the application template class is shown in Figure 4.

As an example, an E-commerce system is modeled as a hierarchical set of services, where the root represents a logical view of the whole electronic commerce system and each node represents a component of the electronic commerce service. Figure 5 shows the E-commerce template model. Each of the nodes represents a component (logical or physical) that should be discovered. Arcs between nodes represent containment relationships between components. Each of the containment relationships has a cardinality requirement. For example, the E-commerce service in the figure consists of one or more Microsoft Internet Information Servers (IIS) and one or more Microsoft SQL servers.

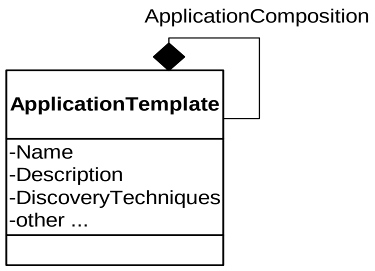


Figure 4: Application template class

An application template object specifies the set of all attributes that have to be discovered in order to identify and manage instances of that application. The *ApplicationComponent* association class in the application template class is used to model the composition relationships between components of an application and the cardinality constraints (if any) of such relationships. We should be able to traverse these relationships in both directions (from children to parent and from parent to children) and are hence modeled in UML as a bi-directional association.

The application instance model consists of an **application instance class** and multiple **application instance objects**. The application instance class defines the standard format for representing the discovered instances of any application. We can think of the discovery engine as a function that maps application template objects to application instance objects. The UML representation for the application instance class is shown in Figure 6

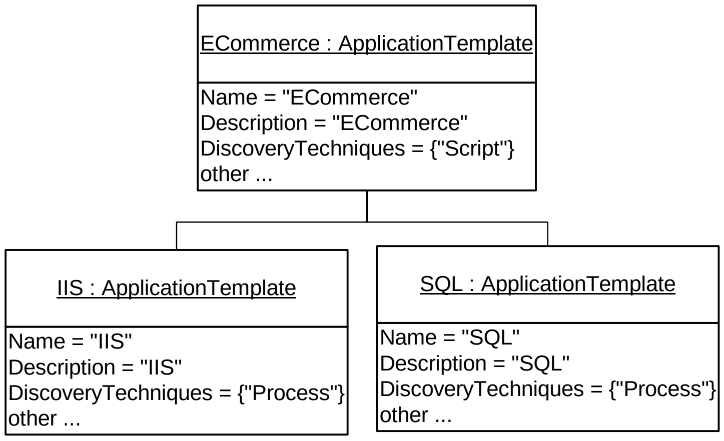

ECommerce : ApplicationTemplate
Name = "ECommerce"
Description = "ECommerce"
DiscoveryTechniques = {"Script"}
other ...
IIS : ApplicationTemplate
Name = "IIS"
Description = "IIS"
DiscoveryTechniques = {"Process"}
other ...
SQL : ApplicationTemplate
Name = "SQL"
Description = "SQL"
DiscoveryTechniques = {"Process"}
other ...

Figure 5: Template model of Microsoft e-commerce

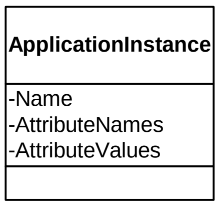


Figure 6: Application instance class

Application template models can be implemented in several ways using existing standards and model repositories. For our implementation, the models are stored in Microsoft's model repository called CIMOM, which is an object manager for the CIM (Common Information Model) standard. In this case, the template models are expressed in the Managed Object Format (MOF) and compiled using the MOF compiler, which parses the models and stores them into CIMOM. The discovery engine can then interface to CIMOM to read the models before or while executing the discovery process. Another option would be represent the models in XML (Extended Modeling Language) and store them in files on a web server. In this case, the discovery engine could connect to the appropriate URL on the web server and read the template models.

4.2 Application Discovery Techniques

At the high level, there are two ways of discovering information about an application: (1) by querying the application directly, or (2) by getting the required information indirectly from other sources. The APIs provided by the application could be used to query the application for information about its attributes and relationships to other components. On the other hand, other applications including the operating systems and middleware provide useful information about the applications that run on them. Some common discovery techniques include:

- **Operating Systems:** Various operating systems' interfaces are used to detect the existence of some applications and to determine their attributes. Although these techniques are usually operating systems specific, some are applicable to multiple operating systems. These techniques include:
    1. Process-based discovery: look for a running process by name or process ID.
    2. File-based discovery: search for a certain file.
    3. Service-based discovery: determine if certain services are installed or active.
    4. Registry-based discovery: look for entries in the MS Windows registry.
    5. Network services: utilize pre-collected information about the network to search for specific services and applications on the network. Examples include the *Windows Browser Service* on Windows NT and *Domain Name Service (DNS)* and *SNMP* on the Internet.

- **Application APIs:** using the APIs provided by the application to determine certain configuration and management attributes. These techniques include:
    1. Scripting: executing pre-defined or customized scripts, which invoke application APIs, to get the required information.
    2. Custom-built executables: using application-specific APIs (e.g., in C,C++) to discover application attributes.

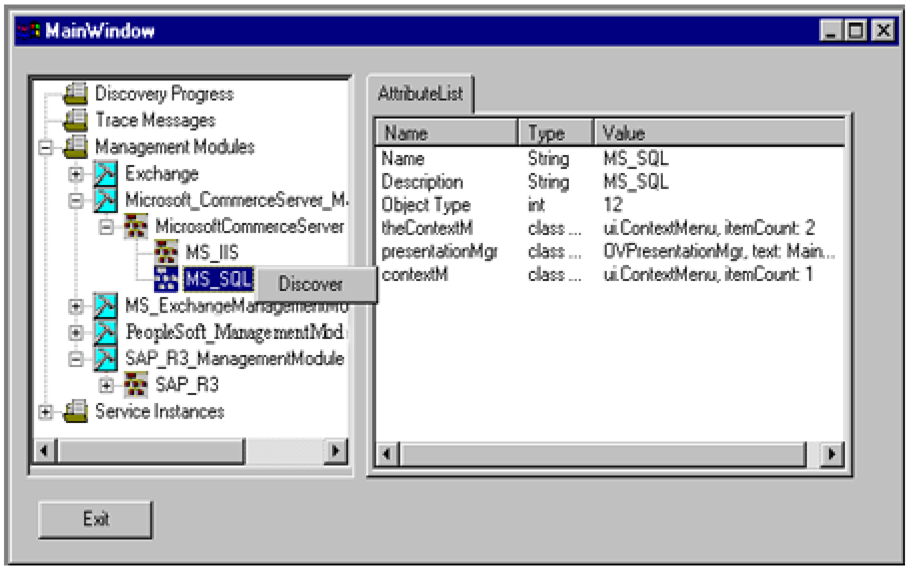
Most discovery techniques can be classified under one of the above categories. Some techniques, however, might be a combination of two or more of the ones mentioned above. The main idea is to have a generic way of representing and executing these techniques, which can be captured in the application template model to drive the discovery. The application template model shown in Figure 8 has an attribute called *DiscoveryTechniques* which is used to model the exact set of techniques used in discovering instances of that application component.

5. Auto-discovery Engine Prototype and Experiments

In this section, we describe a discovery prototype that was built using the concepts described in this report, and discuss the results of two experiments for discovering an e-commerce application and a SAP installation. The prototype is implemented on the Windows NT platform using several programming languages including C++, Java, and VBScript. It implements a model-driven auto-discovery engine that consists of the following main components:

- **Discovery console:** which is used to initiate the discovery process and displays the discovery progress and the discovered instances in real-time (see Figure 7).

- **Discovery policies:** are rules composed of application templates and discovery techniques to be used to locate and identify instances of the required application. These policies are represented in CIM (Common Information Model) and stored in the CIMOM (CIM Object manager).
- **Discovery agents:** are responsible for executing discovery policies and storing the discovered instances in the CIMOM.
- **Publish/subscribe bus:** is the communication protocol between agents. All messages are sent (published) on the bus and participants can listen (subscribe) to the bus and receive only the messages that match their subscription (See [2] for more information about communication middleware alternatives).



Name	Type	Value
Name	String	MS_SQL
Description	String	MS_SQL
Object Type	int	12
theContextM	class ...	ui.ContextMenu, itemCount: 2
presentationMgr	class ...	OVPresentationMgr, text: Main...
contextM	class ...	ui.ContextMenu, itemCount: 1

Figure 7: The discovery console showing several application template models

The discovery process starts by issuing a request from the console to discover a certain application (by selecting the corresponding template from a template tree). The request is sent to all the machines participating in the discovery. Agents on each machine respond to the published request, read the required model from the CIMOM, and execute the corresponding discovery policies. Next, the agents write the discovered instances and their relations in the CIMOM. During the whole process, the discovery console listens to progress messages sent by the agents and provides a real-time visual progress report. As instances of application components are discovered, the results could be used to further scope or narrow the search for other components. The initial scope for the application could be specified as a set of machines or could be modeled as yet another discovery policy.

To validate the applicability and flexibility of our approach, this prototype was used to discover different types of distributed applications. In the following subsections, we describe two of these experiments, an e-commerce application using Microsoft Commerce Server, and a SAP R/3 installation.

5.1 Discovering E-Commerce Application

In general, an e-commerce application consists of one or more web servers connected to one or more database servers. The web server executes the web pages (ASP, HTML, or CGI) containing the business logic. The database server, on the other hand, represents the back-end of the application, where the data needed by the application is stored.

In this example, we used a simple configuration consisting of one web server running Microsoft Internet Information Service (IIS) and one Microsoft SQL Server. The template model used for discovery is shown in Figure 5. The model is represented in MOF and is compiled into a model repository (CIMOM). Part of the MOF file is shown in Figure 8. The discovery process starts by issuing a discovery request from the discovery console, stating the appropriate application template model, as shown in Figure 7.

```
instance of ApplicationTemplate as $IIS
{
      Name = “IIS”;
      Description = “Microsoft Internet Server”;
      DiscoveryTechniques = “process”;
      DiscoveryParameters = “inetinfo.exe”;
      // other ...
};
```

Figure 8: Part of a MOF file showing the application template model of IIS.

This request is published on the bus to be received by the agents on the participating machines. The agent at each machine starts by looking for an IIS server running on that machine using the process-discovery technique. If an IIS is found an instance of an e-commerce node is created and is connected to an instance of IIS. The next step is to find the SQL Servers related to this instance of the application. Finally, the discovered instances and their relationships are stored in a designated CIMOM for later use by the management system.

5.2 Discovering SAP R/3 System

SAP is a different type of distributed application, which consists of several functional modules that could be deployed on multiple machines [6]. SAP is modeled as a hierarchical set of services, where the root represents a logical view of the whole SAP system and each node represents a SAP service. The template model we used to represent a generic SAP system is shown in Figure 9.

The process starts the same way as the previous example. In this case, the user would select the SAP template model from the discovery console to issue a discover request. The only difference between this example and the previous one is the template model used to represent the application and to specify the discovery techniques required for each node. The same discovery engine used without any change. Based on the SAP template model, the discovery process starts by looking for an SAP installation on each machine (registry discovery). Once an installation is found, an API-based technique (wrapped in a script) is used to discover certain parameters of the SAP installation and the result are sent back to the agent.

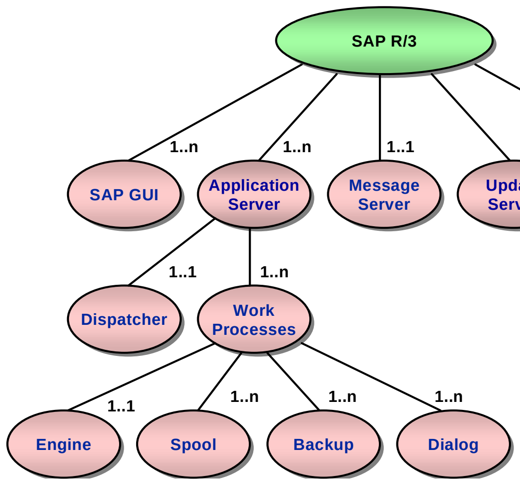


Figure 9: SAP template model

6. Conclusion

In this report, we presented a generic model-based approach to application auto-discovery that can be used to discover different types of distributed applications. Application template models are used to represent the structure and attributes of the application to be discovered and the discovery techniques needed to perform the discovery. The discovery is carried out by discovery agents that execute the appropriate discovery techniques in a certain order specified in the model. A discovery prototype based on the proposed approach was implemented and used to discover several distributed applications. The flexibility and generality of the proposed approach were demonstrated by applying it to discover diverse types of distributed applications including Microsoft e-commerce application and SAP R/3 system. We believe that this approach provides an extensible scheme for developing extensible auto-discovery solutions while reducing development time and effort, and promoting software reuse.

Acknowledgment

We would like to express our thanks and appreciation to Jim Beninga, John Green, John T. Lee, Jim Pruyne, Troy Shahoumian, and Joe Sventek for their valuable input and productive discussions.

References

[1]. G. Booch, J. Rumbaugh, and I. Jacobson, "The unified modeling language user guide", Addison Wesley, 1998.

[2]. J. Colonna-Romano and P. Srite “The middleware source book”, Digital Press, Butterworth-Heinemann.1995

[3]. J. E. Cook and A. L. Wolf, “Automating process discovery through event-data analysis.” ICSE ‘95, pp. 73-82, Seattle, WA 1995.

[4]. Desktop Management Task Force (DMTF), *Common Information Model (CIM)*, http://www.dmtf.org/spec/cims.html, February 1999.

[5]. C Gillooly “Disillusionment in enterprise management”, Information Week, Feb 1998.

[6]. J. A. Hernandez “The SAP R/3 handbook”. McGraw-Hill 1997.

[7]. T. Imielinski and H. Mannila, “A database perspective on knowledge discovery.” Communications of the ACM. Vol. 39 No. 11, pp. 58-64, November 1996.

[8]. H. C. Lin, C. S. Ye, and C. C. Lin, “Automatic topology discovery and virtual connection trace for ATM networks using SNMP.” Proceedings of the Sixth IFIP/IEEE International Symposium on Integrated Network Management, pp. 939-940, Boston, MA. April 1999.

[9]. Microsoft Developer Network, *CIM Object Manager (CIMOM) guide*, 1999.

[10]. C. E. Perkins and H. Harjono, “Resource discovery protocol for mobile computing.” Mobile Networks and Applications. Vol. 1, pp. 447-455. 1996

[11]. S. Ramanathan, D. Caswell, and S. Neal. Personal communication, HP Labs, Palo Alto, CA. 1997

[12]. R. Siamwalla, R. Sharma, and S. Keshav, “Discovering Internet topology”, Technical report, Cornell University. July 1998.

[13]. R. Sturn and W. Winston, "Foundations of application management", John Wiley & Sons, Inc. 1999.

[14]. J. C. Wu “Automatic discovery of network elements,” *U.S. Patent No. 5185860*, Hewlett-Packard Company. May 1990.