

Final thesis


**Automated Network Node Discovery
and Topology Analysis**

by

Johan Sigholm

LITH-IDA-EX--07/061--SE

2007-11-20

 Avdelning, Institution Division, Department Real-Time Systems Laboratory, Dept. of Computer and Information Science 581 83 Linköping		Datum Date 2007-11-20
Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN - <hr/> ISRN LITH-IDA-EX--07/061--SE <hr/> Serietitel och serienummer ISSN Title of series, numbering - <hr/> Linköping Studies in Science and Technology Thesis No. 061
URL för elektronisk version		
Titel Title Automated Network Node Discovery and Topology Analysis Författare Author Johan Sigholm		
Sammanfattning Abstract <p>This Master's Thesis describes the design and development of an architecture for automated network node discovery and topology analysis, implemented as an extension to the network management and provisioning system NETAdmin. The architecture includes functionality for flexible network model assessment, using a method for versatile comparison between off-line database models and real-world models. These models populated by current node data collected by network sensors.</p> <p>The presented architecture supports (1) efficient creation and synchronization of network topology information (2) accurate recognition of new, replaced and upgraded nodes, including rogue nodes that may exhibit malicious behavior, and (3) provides an extension of an existing vendor-neutral enterprise network management and provisioning system.</p> <p>An evaluation of the implementation shows evidence of accurate discovery and classification of unmatched hosts in a live customer production network with over 400 nodes, and presents data on performance and scalability levels.</p> <p>The work was carried out at Netadmin System i Sverige AB, in Linköping, Sweden.</p>		
Nyckelord Keywords Network, discovery, topology, SNMP, XML, regexp		

Final thesis

Automated Network Node Discovery and Topology Analysis

by

Johan Sigholm

LITH-IDA-EX--07/061--SE

2007-11-20

Supervisor: Göran Runfeldt

Examiner: Simin Nadjm-Tehrani

Abstract

This Master's Thesis describes the design and development of an architecture for automated network node discovery and topology analysis, implemented as an extension to the network management and provisioning system NETAdmin. The architecture includes functionality for flexible network model assessment, using a method for versatile comparison between off-line database models and real-world models. These models are populated by current node data collected by network sensors.

The presented architecture supports (1) efficient creation and synchronization of network topology information (2) accurate recognition of new, replaced and upgraded nodes, including rogue nodes that may exhibit malicious behavior, and (3) provides an extension of an existing vendor-neutral enterprise network management and provisioning system.

An evaluation of the implementation shows evidence of accurate discovery and classification of unmatched hosts in a live customer production network with over 400 nodes, and presents data on performance and scalability levels.

The work was carried out at Netadmin System i Sverige AB, in Linköping, Sweden.



Acknowledgements

During this Master's Thesis project I have continuously received valuable feedback and guidance from my examiner Simin Nadjm-Tehrani, which is gratefully acknowledged.

I would also like to thank my supervisor at Netadmin System i Sverige AB, Göran Runfeldt, for giving me crucial support and advice during the implementation and testing stages of this project.

Finally, I wish to express a word of gratitude to all other co-workers at Netadmin, who have helped me in many ways during my time there.



Contents

1	Introduction	1
1.1	Background	1
1.2	Netadmin company history	2
1.3	Problem description	3
1.4	Objective	4
1.5	Approach	4
1.6	Limitations	6
1.7	Related work	6
1.8	Document information	7
1.8.1	Document overview	7
1.8.2	Reading instructions	7
1.8.3	Withheld sections	8
1.9	Publication	8
2	The NETAdmin system	9
2.1	Main features	9
2.1.1	Automatic service provisioning	9
2.1.2	Tiered design	10
2.1.3	Other features	10
2.2	Technical design	10
2.3	Discovery service extensions	11
3	Network discovery	13
3.1	Purpose	13
3.2	Levels of discovery	13
3.3	Topology changes	14
3.4	Vendor dependency	14
3.5	Protocols	14
3.5.1	Simple Network Management Protocol	14
3.5.2	Link Layer Discovery Protocol	15
3.5.3	Vendor-dependent protocols	15
3.6	Automatic node discovery	15
3.6.1	Data link layer discovery	16
3.6.2	Network layer discovery	16

3.7	Discovery approaches	16
4	Discovery Agent	17
4.1	Design	17
4.1.1	Programming language	17
4.1.2	Design tools	18
4.2	Implementation	19
4.3	Networks processor	19
4.4	Network scanner	19
4.4.1	Scanning approaches	19
4.4.2	Scanning tools	21
4.5	Host scanner	21
4.5.1	Communities	22
4.5.2	Host signature	22
4.6	Performance issues	22
4.6.1	Parallelization	23
4.6.2	Resource scheduling	23
5	Topology Analysis Engine	25
5.1	Design	25
5.2	Languages	25
5.2.1	The .NET framework	25
5.2.2	Visual Basic .NET	26
5.2.3	Design tools	26
5.3	Implementation	26
5.3.1	Initialization	26
5.3.2	Host processing	27
5.3.3	Signature analysis	28
5.3.4	Host identification	28
5.3.5	Classifying unknowns	28
5.3.6	Cycle completion	29
6	User Interface	31
6.1	Design	31
6.1.1	Languages	31
6.2	Implementation	31
6.2.1	General settings	31
6.2.2	Detailed configuration	32
6.2.3	Storing changes	32
6.2.4	The discovery log	32
6.2.5	Access control	33

7	Testing and evaluation	35
7.1	Test environments	35
7.1.1	Staging environment tests	35
7.1.2	Live tests	36
7.2	Performance evaluation	36
7.2.1	Top-down implementation	36
7.2.2	Parallelized implementation	36
7.2.3	Scheduling evaluation	37
7.2.4	Evaluation environment	38
7.2.5	Results	38
8	Discussion	39
8.1	Results	39
8.2	Future work	39
8.3	Conclusions	40
	Glossary	41
A	Screenshots	47
	PHPEclipse developing environment	48
	MySQL Control Center	49
	Visual Basic .NET environment	50
	Visual InterDev environment	51
	General network settings	52
	Detailed discovery settings	53
	Network discovery log	54
	Bibliography	55
	Index	57

Chapter 1

Introduction

This chapter describes the background and problem formulation for the Master's Thesis project, which is submitted as a partial fulfillment for the degree of Master of Science in Computer Science and Engineering at Linköping University.

1.1 Background

The NETAdmin system, developed by the Swedish company Netadmin System i Sverige AB, helps network owners and service providers deal with the problems of efficient network management and service provisioning. The main goal of the system is to automate and facilitate repetitive activities, which have traditionally required manual interaction. This includes administrative tasks such as adding and removing end-users, handling end-user support issues, and sending out bills to the end-users depending on what services they are subscribing to. It also includes technical tasks such as configuring network hardware, monitoring the network health and the gathering of statistics.

After installation in a network, the system works as an intelligent and integrated component, which can be remotely accessed and managed. It can then control many parts of the network, such as configuring the active network equipment (e.g. switches, routers or servers) to make sure each end-user gets the service that he or she is subscribing to.

Figure 1.1 gives an idea of what the role of the NETAdmin system is in a network. It controls the network infrastructure in order to provision services offered by different providers, but another important function is to act as a seamless link between the service provider and the end-user, making the intermediate infrastructure invisible. This allows for effective and healthy competition between many providers in a shared network, while at the same time allowing the network operator to focus on the management of the network.

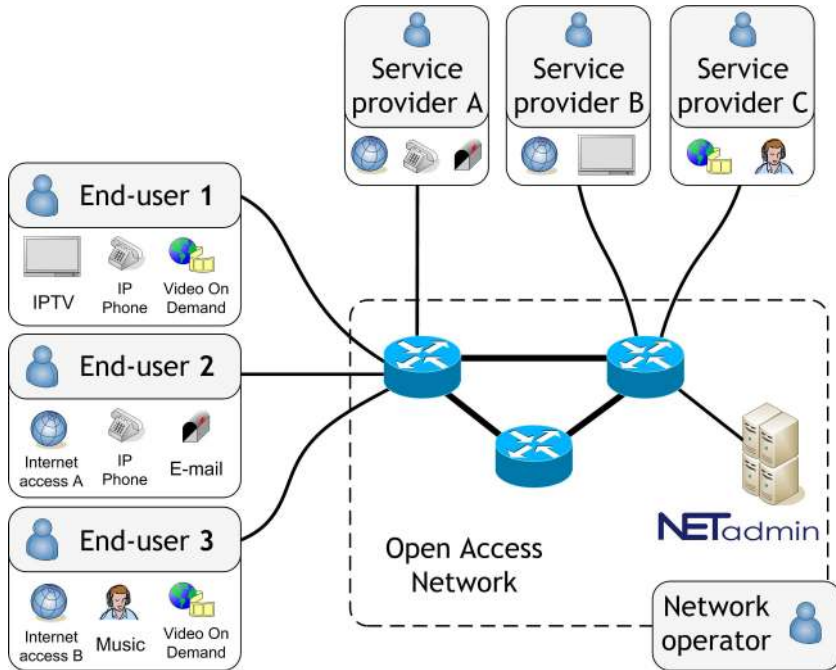


Figure 1.1: The NETAdmin system provisions services offered by service providers to the end-users. By automating configuration of the network infrastructure, the work of the network operator is greatly facilitated.

It should be noted that when the term “customer” is used in this document, it refers to customers of Netadmin System i Sverige AB, i.e. the network operators or service providers running the NETAdmin system. The other plausible implication of a customer, as a client of a service provider in a given network, is instead referred to as an “end-user”.

The NETAdmin system is deployed in a wide variety of networks in Europe and North America, but has its largest customer base in Scandinavia. In Sweden, many large network operators, such as TeliaSonera, and a large part of the metropolitan area networks, such as Gothnet in Göteborg and Utsikt in Linköping, currently use NETAdmin in their production networks.

1.2 Netadmin company history

The development of the NETAdmin system began in 1998 as a project within the Vadstena-based company Wasadata System AB. Wasadata’s broadband network was growing rapidly at the time, and there was a need for a tool to facilitate the process of setting up and managing network equipment,

mainly switches, and handling end-user data. Hence, this was the goal of the NETadmin system, which was launched for the first time in 1999.

After a few years it was decided that the system itself might be valuable to other companies in need of a solution for network management. In 2002 the decision was made to turn NETadmin into an independent product. Shortly after this the system was installed in the first customer networks, Älvsbyns kommun and Calypso Internetservice AB. During 2003 and 2004 more resources were allocated to NETadmin and in December 2004 the independent company Netadmin System i Sverige AB was founded, as a subsidiary of Wasadata System AB.

Today Netadmin's main customers are network owners, Internet service providers, regional energy companies and metropolitan area network operators. Netadmin also has international customers, in both Europe and North America, and is presently in a phase of rapid growth.

1.3 Problem description

In small computer networks, such as those found in small to medium-sized businesses, management can usually be handled manually by a network administrator. However, as networks grow larger, managing clients, servers, and other network elements can become a time-consuming task. In these networks it is necessary to employ an automatic tool to support handling of changes to the network in an efficient manner.

Furthermore, computer networks are rarely static; rather, dynamic changes are a constant feature of enterprise networks. New nodes are continuously added, removed or updated with new configurations. These changes can be in the form of a completely new device being installed in the network (such as a switch or a router), but also an existing device being upgraded with a new interface module, line card, power supply etc. Keeping track of these changes manually, and making appropriate updates, may take a lot of time and effort.

Where a management system is deployed to support these activities, the technician performing hardware installations is usually not the same person as the one in charge of entering the new configuration information into the management system. This is especially true when several different companies are involved in the operation of the network, such as in the case of operator neutral Open Access Networks (OANs) [1].

OANs have become increasingly popular over the last few years. For example, in Sweden this has virtually become a norm when building new networks, partly because of a high level of government co-funding, but also because of requirements imposed by the regulatory authorities [2]. In these networks the end-users, usually connected by use of broadband techniques, can choose between several competing service providers delivering general Internet access, IP telephony, IPTV, video on demand or other media services.

In order to come to terms with the problems of changes being made to the network equipment without the proper corresponding updates in the NETAdmin system, a method for network node discovery and topology analysis was needed. The NETAdmin system was also in need of functionality to handle discovery of new network nodes, to recognize equipment by vendor and identify changes in the network topology. This should be done in an automated fashion which minimizes the need of manual human interaction.

1.4 Objective

This thesis should help answer the following questions:

- How can the desired functionality for network discovery and topology analysis be added to the NETAdmin system?
- How well does the proposed implementation work in simulated and production environments?
- What is the performance/scalability of the implementation in shared environments?
- How can the implementation be further developed in the future?

The objective of the project is thus to implement new functionality in the NETAdmin system for automated discovery of network nodes with detection of configuration changes and generation of network topology.

1.5 Approach

After an initial evaluation of the presented problems a high-level architecture was designed to serve as a base for the forthcoming implementation. The ambition was to make the architecture as uncomplicated and straightforward as possible, allowing for a transparent and flexible implementation. The suggested approach is to create three main components; a Discovery Agent, a Topology Analysis Engine, and a User Interface. Figure 1.2 gives a high-level description of the proposed architecture.

1. The network administrator configures the network which should be scanned through the User Interface. The request is sent out to a Discovery Agent.
2. The Discovery Agent scans the network and all active nodes, according to the received configuration. A signature is created for each discovered node, containing its characteristics.
3. The signatures are transferred to the Topology Analysis Engine.

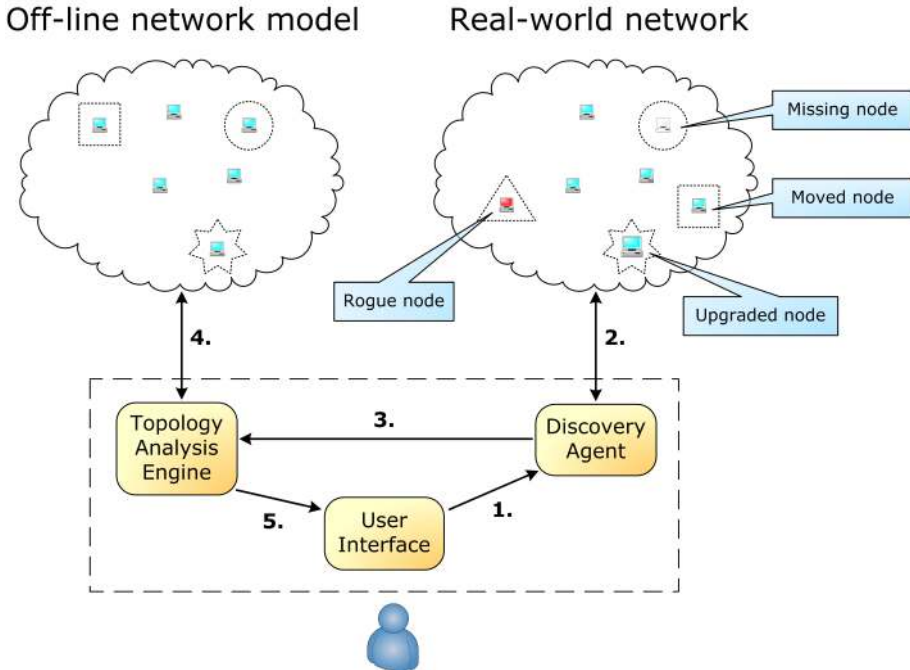


Figure 1.2: High-level discovery and topology analysis architecture

4. The information is processed by comparing the live results with the off-line model database.
5. A digest list of conclusions is presented to the network administrator via the User Interface, who can then act upon this information.

By using this basic architecture, a high level of flexibility is attained. The network administrator may choose exactly which node information that is of interest, for example IP or MAC addresses, system names, software versions, hardware modules installed or active running services. Standard tools to retrieve this information are implemented at the Discovery Agents. The resulting data is stored in XML signature files, which can be parsed by the Topology Analysis Engine.

The main tasks which should be handled effectively by an implementation of this architecture are:

- Facilitating the first-time creation of the off-line network model, by automating the discovery process.
- Continuous monitoring of changes in the network, which are overlooked due to either a failure of communication between responsible people or by pure negligence.

- Detection of unknown, possibly malicious network elements, in order to minimize the inflicted damage.

1.6 Limitations

Effective automation of repetitive tasks while maintaining application constraints is a non-trivial task, also when it comes to discovery of network information. The prospect of being able to let machines handle information gathering and processing without any involvement of human beings is desirable since money can be saved. A higher grade of automation does, however, require more care since it is hard to program computers to always make the right decision. The solution presented in this thesis will require human interaction through a network administrator, to decide if a certain action should be taken or not. It will also focus on identification using standard protocols, with the possibility for future extension with vendor-dependent protocols.

Another limitation regarding the topology analysis implementation in this thesis is that the focus has been put on discovery of the network nodes (active network elements), and not the edges (links between the nodes). An extension for link discovery is however supported by the general architecture design, and is suggested by the author as a future improvement.

Finally, the discovery implementation has focused on active network scanning rather than passive discovery. An explanation of the differences between these methods and the reasoning behind the decision of favoring active scanning can be found in Chapter 3.

1.7 Related work

Several research projects have identified the need for a topology agent where information about the hosts, their software patch levels, and known vulnerabilities are correlated to the alerts produced by the infosec devices in order to eliminate or aggregate some alerts [3], or to facilitate software updates [4]. Designing discovery services to aid decision-plane algorithms, maintaining correct semantics during low-level network change and allowing zero pre-configuration has also been identified as an important objective when constructing future network architectures [5].

There have also been projects aimed at designing tools for both vendor-dependent and independent topology discovery [6]. The NetMap project [7] provides a solution for the use of a variety of composable network tools, each used for a specific purpose. There are also many available software solutions for network management and security monitoring, both open source projects like Netdisco [8] and Nessus [9], and proprietary solutions like HP OpenView [10] and Microsoft SMS [11]. However, these tools do not supply functionality for discrepancy detection or the possibility to practically compare two network models in an enterprise environment.

1.8 Document information

This section contains information about the thesis document, including an overview of the document, reading instructions, and information about certain withheld sections.

1.8.1 Document overview

This document comprises the following chapters:

- | | |
|------------------------------------|---|
| 1. Introduction | This chapter includes a description of the background of the project and Netadmin company, and an introduction to the problem and selected method and approach. |
| 2. The NETadmin system | A description of the NETadmin system, the different parts and their functionalities. |
| 3. Network discovery | This chapter gives an introduction to network discovery. |
| 4. Discovery Agent | The Discovery Agent implementation is presented in this chapter. |
| 5. Topology Analysis Engine | A description of the implementation of the Topology Analysis Engine and the process of rule matching is presented in this chapter. |
| 6. User Interface | This chapter gives information about how the administration and configuration of the discovery system is performed via the NETadmin graphical user interface. |
| 7. Testing and evaluation | In this chapter results of both testing and evaluation of the complete project are given. |
| 8. Discussion | Some reflections of the results of the Master's Thesis project are presented along with suggestions of future improvements. |

1.8.2 Reading instructions

Chapter 2 is mainly valuable for readers not previously familiar with the NETadmin system and its features and functionality. If the reader is familiar with how network discovery works, Chapter 3 may be also safely skipped. Many of the terms and abbreviations which appear throughout this thesis are explained in detail in the Glossary, which can be used as a quick reference. An index is also available at the end of the document.

1.8.3 Withheld sections

Since this Master's Thesis was carried out at a commercial company, some parts of the project cannot be disclosed in this report due to confidentiality and copyright issues. This includes details concerning database design (ER-diagrams, relational schemas and table layout), and the project source code (except for some selected samples shown in the Appendix).

1.9 Publication

A summary of the work performed in this Master's Thesis project, along with the main results, was presented in a paper at the 2nd Benelux Workshop on Information and System Security (WISSec2007), Luxembourg City, Luxembourg, September 20-21, 2007. The paper had the following title: Sigholm J., Nadjm-Tehrani S., "Enterprise Network Node Discovery and Topology Analysis: An Experience Report".

Chapter 2

The NETadmin system

NETadmin is a complex system built up of many parts. This chapter will describe the main features of the system, the general design, and aspects of the various system modules and how they interact with each other.

2.1 Main features

2.1.1 Automatic service provisioning

One of the main features of the NETadmin system is that it provides fully automatic service provisioning. This means that much of the work that traditionally was performed by the customer services department, such as new signups, requests of service changes, and subscription cancellations, has been transferred to the end-user.

This is done by adding a so-called “captive portal”, through which the end-users can select which services they want to subscribe to, or cancel, without having to call customer services and talk to a human being. When an end-user plugs in his or her computer to the network for the first time, and starts up a web browser, this will bring up the captive portal home page, where the end-user may select which services and which service providers that are desired.

Captive portals are commonly used in shared access environments, such as university campus networks, where authentication is required. They are also useful in situations where specific information is needed for billing purposes, such as in commercial wireless networks at hotels or Internet cafés. Any client using the HTTP protocol to access a web page is forced to go to a certain predestined location containing the portal page.

The most common techniques used to achieve this is by either letting the DNS server reply with the captive portals IP address for any query received from an unauthorized client, or by using a firewall to forward all HTTP traffic to a certain server containing the portal home page. In the

NETAdmin system the DNS implementation is used.

2.1.2 Tiered design

Another major NETAdmin feature is the tiered system design, which separates the network owner/operator and the service providers. This works by letting each party have its own separate NETAdmin installation, with only the information that is associated with their business, such as list of their end-users, billing information, IP-addresses etc. Each end-user can also log in to the captive portal and view or update their personal information or information regarding their current selected services.

Each service provider's NETAdmin system is connected to the network operator's NETAdmin system, and information is exchanged securely between them. An example of this is when an end-user signs up for a certain service, either through the captive portal or by calling in, this information is synchronized between the service provider's and the network operator's NETAdmin systems. The same method is used to exchange information regarding support, abuse or billing issues which needs to be handled by several parties.

In this way many service providers can be present in the network, as is often the case in Open Access Networks, while confidential business information, such as end-user information, is not visible to competitors. Certain information can however, at the same time, be shared between a service provider and the network operator when this is needed. This allows for the effective competition which is usually desired, while still maintaining flexible information flows.

2.1.3 Other features

The NETAdmin system also contains a variety of other features, such as a comprehensive case system (for handling e.g. support, billing or abuse cases), an advanced network monitoring system with statistics and fault alarm capabilities, and a user management system which can link end-users to relevant cases. These features will however not be discussed in further detail in this thesis.

2.2 Technical design

The NETAdmin system is based on two main hardware platforms, Windows 2003 Server and NETserver, a NETAdmin-customized version of Debian GNU/Linux. The NETservers take care of tasks like network hardware configuration, resource provisioning, gathering of statistics and monitoring. The Windows servers act both as a front-end, with graphical user interfaces, and as a logical processing unit which sends out "work orders" to the various NETservers.

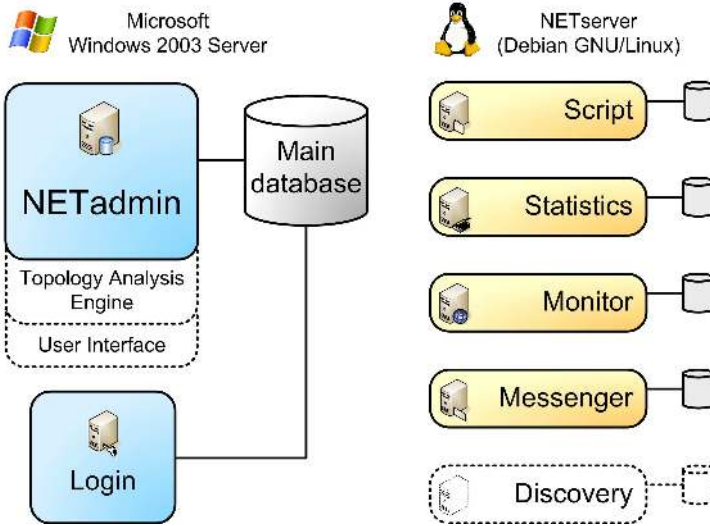


Figure 2.1: The NETAdmin system with proposed extensions

Data storage is implemented by use of databases. Currently MySQL databases are used on both Windows and Linux platforms. The main NETAdmin database is hosted on the Windows server and contains all main information about the system, settings, end-user data etc. The NETservers usually have their own databases, mainly used for temporary storage of collected network data which is to be synchronized with the main database. Communication between the servers is done by either secure HTTPS connections or by SSL-encrypted SQL queries directly to the common databases.

2.3 Discovery service extensions

Figure 2.1 shows an overview of the NETAdmin system components with the proposed new modules as dotted elements. For the network discovery implementation, a new NETserver, including a local database, is added, which acts as the Discovery Agent.

The Topology Analysis Engine is implemented as a “runner” in the NETAdmin domain. This is an extension of the functionality in the main Windows server, with a logical module for the analysis process. Its task is to compare the data collected by the discovery NETserver, and decide if any changes have occurred and if so what exactly has happened.

For the configuration and management of the discovery system an extension of the existing NETAdmin graphical user interface (GUI) is done. The User Interface module also contains functionality for sending out updated configuration setting to the Discovery Agents.

Chapter 3

Network discovery

This chapter will give an introduction to network discovery, its purpose, relevance, and various aspects on implementation.

3.1 Purpose

The goal of network discovery is to facilitate the management of large or complex computer networks, where it is difficult to manually keep track of the topology. By having access to an up to date view of the network, decisions about hardware upgrades, software updates or capacity improvements can more easily be made. Knowing what equipment that is attached to the network is also important from an information security perspective. This helps the network administrator verify that the discovered nodes are indeed authorized, and that the equipment is updated with the most recent software or patch levels.

3.2 Levels of discovery

Network discovery can be performed on many different levels, and with different granularity. The preferred level of discovery can correspond to a certain layer in the OSI reference model [12] (e.g. physical, data link or network layer), it can refer to targeting on a certain part of a larger network, or focusing on a certain type of network (e.g. fixed access network, wireless network, etc.).

What level of discovery is appropriate may vary depending on the purpose of the network discovery and which degree of detail is desired. A high level of abstraction might be ideal when seeking a general overview of the network, while a finer granularity could be necessary in cases where a complete map of the network topology is desired.

3.3 Topology changes

As previously mentioned most networks are in a constant state of change, nodes will continuously join or leave the network, and new links may appear or vanish. Dealing with these variations in the network topology is a challenging part of the discovery process, and any discovery implementation must take this into consideration.

3.4 Vendor dependency

Computer networks with components from a single vendor, based on proprietary hardware and software, is preferred in some networks where interoperability guarantees are needed. In these networks discovery is relatively straight forward, due to the fact that the protocols and implementations are known. In smaller enterprise networks vendor-dependent designs might be more common, but in larger networks you often find equipment from several vendors.

Designing Network discovery for heterogeneous networks requires a vendor-neutral approach. This involves using standard protocols, or supporting protocols from many different vendors. This is very important, since the effectiveness of a discovery service is dependant on what data can be gathered from the attached nodes in the network.

3.5 Protocols

Being able to discover the topology of a network demands that you can successfully communicate with the network elements. Communication is done via protocols common to both the elements and the discovering client.

3.5.1 Simple Network Management Protocol

The Simple Network Management Protocol or SNMP is a part of the TCP/IP protocol suite defined by the Internet Engineering Task Force (IETF). It is an application layer protocol used for communication with network devices, to collect information or transmit configuration. SNMP is a very commonly used protocol and is implemented in most available network equipment (e.g. switches, routers, and servers).

Authentication in SNMP is handled by assigning special communities for read and write access, where a community string works like a password. When an authenticated management client has connected to the SNMP enabled device, it can retrieve management information stored as objects, arranged in a tree-like hierarchy. Each object within a sub tree can be identified by a unique number, called Object Identifier (OID). Translations between OIDs and their common names are provided by Management Information Bases (MIBs). There are standard MIBs containing the most

common management information, but there are also special purpose MIBs. By creating and publishing their own MIBs, vendors can extend the standard MIBs with support for their own hardware.

Because the SNMP protocol is vendor-neutral and widely deployed, it is very useful for network discovery. There are a few different versions of the protocol (versions 1 to 3), and while these are not mutually compatible, most SNMP implementations support multiple versions. One limitation of the SNMP protocol is that it only works on the network layer, meaning that the network address (IP-address) for a device must be known in order to contact that unit.

3.5.2 Link Layer Discovery Protocol

The Link Layer Discovery Protocol or LLDP (IEEE 802.1AB) is a vendor-neutral discovery protocol which, as suggested by its name, was designed for efficient data link layer discovery. This protocol gives network devices the ability to identify themselves and their capabilities in the network, but since it was ratified relatively recently (May 2005), much of the equipment in existing networks do not yet have support for this protocol.

3.5.3 Vendor-dependent protocols

There exists a variety of vendor-dependent protocols for networks discovery, such as the Cisco Discovery Protocol (CDP), the Extreme Discovery Protocol (EDP) the Digital Maintenance Operation Protocol (DEC MOP) etc. These protocols have the benefit of being able to communicate on the data link layer, which allows communication without knowledge of network addresses, and between systems supporting different network layer protocols.

A drawback with vendor-dependent protocols is that they only work with equipment from that specific vendor. Additionally problems may arise if equipment from different vendors is mixed, which may lead to wrong topology information being transmitted.

3.6 Automatic node discovery

Being able to automate network discovery and topology analysis is a highly desired feature, since the process can be both time-consuming, and needs to be constantly ongoing due to changes in the network. There is therefore a need of a programmable unit which is responsible for discovering the network and collecting topology information. This unit should be able to perform an unbiased scan of the network and collect the necessary information, with no or only limited prior knowledge of the topology.

3.6.1 Data link layer discovery

If there is no knowledge of what network addresses that are used in the network which should be scanned, the discovery process must be carried out on the data link layer. This can be done by implementing discovery methods such as LLDP or proprietary discovery methods. In homogenous networks this method can work well, but problems may arise in networks with mixed vendors or with equipment that does not support the used data link layer discovery protocol.

3.6.2 Network layer discovery

When the network address and mask are known, discovery of network topology on the network layer is easier. It does however require a scanning process, to decide if each address in the network corresponds to an active node. This process can be implemented as either a passive or an active mapping of the network.

3.7 Discovery approaches

Two major paradigms of network scanning are active and passive scanning. While active scanning tries to detect hosts by actively sending out probes and expecting a reply. Passive scanning, on the other hand, merely listens to traffic on the network to detect specific hosts.

Which technique that is the best to use in a given situation may vary depending on what the conditions are and what results you are hoping to achieve. In a properly configured network all hosts should reply to an active request for a reply. In networks which include hosts that are misconfigured or do not reply to echo requests due to firewall or policy rules, a passive scanning method might be the only option to detect hosts.

Chapter 4

Discovery Agent

4.1 Design

The design of the Discovery Agent was initially decided to be based on the traditional procedural programming paradigm, with a top-down approach, without any object orientation techniques. The reason for this was that the source code was not expected to be especially complex or substantial. Many of the functions previously implemented on NETservers also used this approach.

The design of the discovery database was done after an evaluation of the information that was desired, through an entity-relationship modeling process. It was clear from the beginning that the design had to allow for future extensions, which was why the XML data type was chosen for describing capabilities of individual hosts. Other data types were chosen to match the ones already in use in other NETadmin databases.

4.1.1 Programming language

An evaluation of suitable programming languages was performed in order to decide which one to use for the Discovery Agent implementation. The evaluated languages consisted of Java and C++, Perl, Ruby and PHP.

At first Java or C++ were favored since I personally had prior experience of these languages from other university courses. Both these languages have support for object oriented design. Compiled C++ programs are usually very fast and efficient, while Java programs may be used on many different platforms. These benefits were however not very relevant in the current situation, as a non-object oriented design method had been chosen, and the NETserver platform would remain constant.

Since most of the resource demanding functionality was expected to be performed by external tools, a scripting language was deemed to be more suitable for the implementation. The Perl, Ruby and PHP languages were

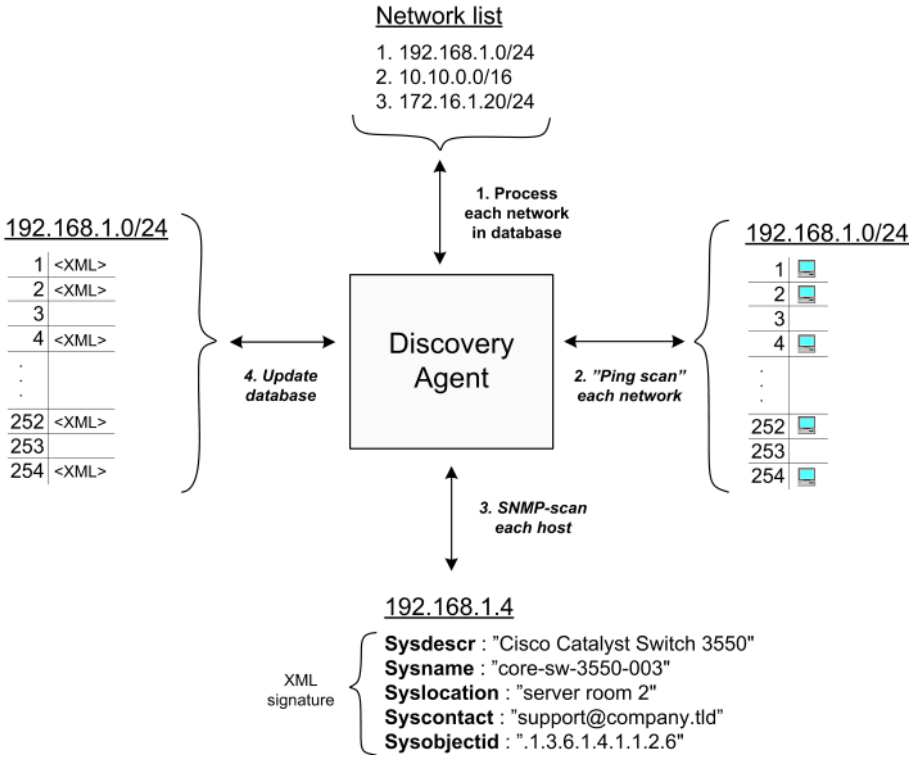


Figure 4.1: The discovery process

considered, where PHP was finally selected due to both having built-in functionality for easy database access, but also because this was the language used by most of the previously existing NETserver modules. I did not have any personal experience of this language prior to this project, but decided that sufficient support should be available through my supervisor and other employees at Netadmin. Some degree of modularity could also be attained by the use of PHP include files, where specific functions may be reused by other scripts.

4.1.2 Design tools

For the development of the source code the Eclipse environment was selected, with the PHPEclipse plug-in [13]. This allowed for easy access to help commands, syntax highlighting, and debugging support. For a screenshot of PHPEclipse environment, see Figure A.1 in the Appendix.

For design and configuration of the MySQL database, the MySQL Control Center (MySQLCC) was used. It is a graphical front-end which allows

for easy database and table creation, and evaluation of SQL queries. For a screenshot of the MySQLCC environment, see Figure A.2.

4.2 Implementation

The Discovery Agent is implemented on a NETserver and is the active data gathering component of the discovery system. The discovery process is illustrated in Figure 4.1.

The discovery system consists of three major parts, the networks processing script, the net scanner script and host scanner script. These scripts are individual scripts, run in a sequential top-down fashion where the networks processing script spawns instances of the net scanner, which in turn spawns instances of the host scanner. This is done in order to achieve the degree of parallelization needed to complete the tasks in a limited timeframe. The results both of single and multi-process scanning are discussed in the testing Chapter 7. See also the UML activity diagram in Figure 4.2 for an illustration of the workflow behavior of the discovery script server.

4.3 Networks processor

This script connects to the discovery script database and retrieves the main information about the networks which are to be scanned. The information is updated by the NETadmin server when a user changes the settings in the User Interface (see Chapter 6). Each network entry in the database is processed to see if it is time to perform a new scan of the current network. This is controlled by the `netperiod` and `netlastscandatetime` fields, which tell how often the net should be scanned (in seconds) and when the last scan was performed.

With these variables, in combination with the current date and time, a decision on whether the net needs to be rescanned can be made. If it is time for a new scan of the current network, a net scanner process is spawned.

4.4 Network scanner

The net scanner script scans a network, designated by a network address and a netmask, to detect which attached hosts that are active. Since there are several possible methods of scanning that can be utilized, a decision had to be made on which to use in this script.

4.4.1 Scanning approaches

As mentioned in section 3.7, two major ways of network scanning are active and passive scanning, where active scanning tries to detect hosts by active probing [14] and passive scanning listens to detect traffic generated by the

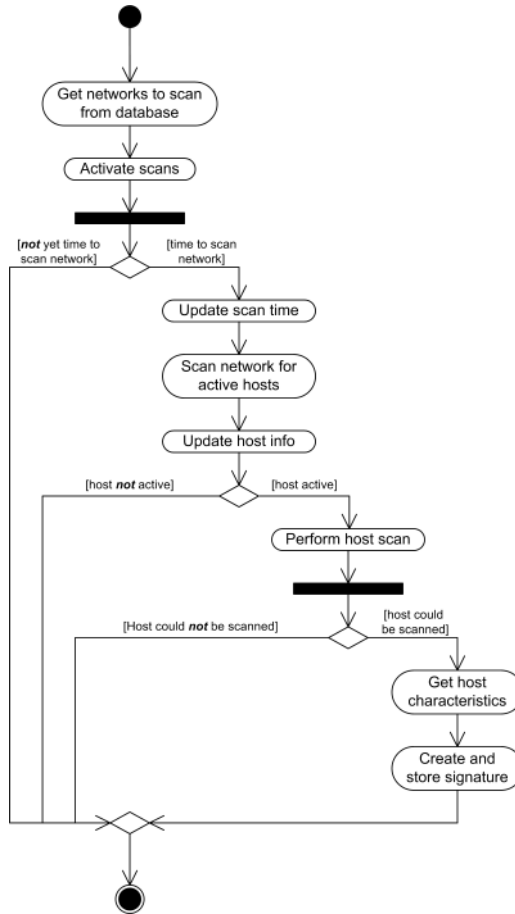


Figure 4.2: Discovery service UML activity diagram

hosts [15]. In the current instance of the architecture active scanning has been favoured since the devices we primarily want to detect (i.e. switches, routers etc.) very seldom generate any unprovoked traffic.

Active scanning is performed by sending out data on the network targeted at the address which you want to detect. What kind of data that you send can vary, but a common and straightforward approach is to send an ICMP (Internet Control Message Protocol) ECHO request. This process is commonly called “pinging” a host, from the name of a tool used to send these messages. One problem with using the ICMP protocol is that not all hosts will reply to it, due to firewalls blocking this type of traffic or giving it a low priority.

Other possible ways to actively scan for hosts are by using TCP or UDP

datagrams, directed towards specific addresses in a given network, in an attempt to generate a response. However, since the aim of the script is to detect hosts with standard configuration, and since the script must be able to scan a large number of hosts in a limited time, it was decided that a regular ICMP ECHO scan would suffice.

4.4.2 Scanning tools

There are several available tools available for performing network scans on the Linux platform. Two of these tools were evaluated to see which would be most suitable, Fping and Nmap.

Fping

Fping [16] is a tool which uses the ICMP protocol to determine if a host is active or not. Fping has a feature which allows the user to specify a range of hosts to be scanned, and is especially designed to be used in scripts, with easily parseable output. Instead of trying one host until it timeouts or replies, Fping will send out a ping packet and move on to the next host in a round-robin fashion. If a host replies, it is noted and removed from the list of hosts to check. If a host does not respond within a certain time limit and/or retry limit it will be considered unreachable.

Nmap

Nmap (Network Mapper) [17] is a free open source utility for network exploration and security auditing. Apart from containing the same functionality as Fping, it also has many advanced scanning features such as support for detecting system services, operating system and MAC and DNS addresses. The results can be presented in an array which is easy to parse.

A performance evaluation showed that using the Nmap program was the most efficient of the available scanning tools and was most suitable for our purposes.

4.5 Host scanner

After a scan of a particular network has been completed, the list of detected active hosts is processed to establish which hosts are new and which have been seen before. This is determined by a combination of IP address, MAC address and a status flag in the database. Thus, a host which was previously inactive (e.g. turned off for a period of time), is considered new if it becomes active again. All new hosts are then further examined individually by the host scanner script.

This module performs the information gathering from the individual active hosts discovered by the network scanner. The main protocol used is

Table 4.1: XML signature for a discovered host

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<signature>
  <sysdescr>Summit11Tx - Version 7.4.2 (Build 6)</sysdescr>
  <sysname>core-sw-01</sysname>
  <syslocation>server room 3</syslocation>
  <syscontact>support@extremenetworks.com</syscontact>
  <sysobjectid>.1.3.6.1.4.1.1916.2.14</sysobjectid>
</signature>
```

the Simple Network Management Protocol (SNMP), which is implemented in most manageable network equipment. More information about the hosts (e.g. available services or running operating system) can also be retrieved by using either Nmap, or more specialized tools such as Xprobe [18].

4.5.1 Communities

To be able to read information from the nodes via SNMP, an SNMP community string is required, which serves as a password for access control. The community strings to be tried are provided by the NETadmin server. They are ultimately based on information entered by the system administrator when configuring the system. The communities can be stored either in plain text or encrypted (if using SNMPv3). In order to optimize the scan speed, a working SNMP community, if found, is stored in the host entry to be tried first the next time.

4.5.2 Host signature

In the basic setting, some standard SNMP information is gathered from the host: *system description*, *system name*, *system location*, *system contact* and *system object ID*. Which information that is gathered can however be dynamically configured. Based on this information, a compound XML host signature is generated (see Table 4.1) and stored in the database for use by the later analysis process by the Topology Analysis Engine.

4.6 Performance issues

In the first implementation of the Discovery Agent, all scans of the networks assigned to an individual agent were done in a serial fashion without any concerns about scalability. Initially this did not cause any problems, but when the implementation was tested in a simulated large network, performance issues arose. A scan of a network with a thousand hosts could take an

hour to complete. This might not seem too bad at first glance, but in even larger networks the scanning times increased linearly. Due to the fact that network topology changes may occur quite frequently, better performance is needed to capture these changes more quickly.

4.6.1 Parallelization

The above mentioned performance requirements demanded a redesign of the scanning implementation. Instead of using a traditional top-down scanning method, a parallel approach was needed. This was implemented by dividing the agent into separate modules which could be run simultaneously.

By scanning several large networks and their respective hosts in parallel, it was shown that completion times could be significantly reduced. More information about the performance tests can be found in section 7.2.

4.6.2 Resource scheduling

The parallelized approach did however require more system resources, in form of primarily CPU and database loads. This led to the problem of dividing system resources of the NETserver between the instances of the scanning modules, and any other programs which may be running on the same server.

Without any resource scheduling the NETserver which was used in the testing environment froze up, or sometimes even crashed, due to the high number of scanning processes being spawned. This was solved by implementing a scheduling function which distributed the scanning evenly over the allotted scan time. Some extra checks were also implemented to make sure that there were enough database connections left free, and that a scan of an individual discovered host would not be started if there was already one running.

Chapter 5

Topology Analysis Engine

5.1 Design

The responsibility of the Topology Analysis Engine is to retrieve the data collected by the Discovery Agent, to process it and finally store it as useful information which can be presented to the user. The runner compares the model of the network which exists in the NETadmin internal database with the real world data gathered by the NETserver, and alerts the user of inconsistencies which might be important to attend to.

The Topology Analysis Engine is designed as a runner in the NETadmin domain – a program scheduled to run constantly in the background. Since there was already an existing underlying framework for creating a runner, through which the new functionality was to be implemented, many design decisions were dependent on compatibility with the existing system.

5.2 Languages

The NETadmin applications running in the Windows 2003 Server environment are developed for .NET, a software framework provided by Microsoft.

5.2.1 The .NET framework

The goal of .NET is to facilitate the development of Windows applications and to reduce the security vulnerability of these applications and the computers they are running on. The .NET framework also contains a large body of pre-coded software solutions to common program requirements, such as data management, database connectivity and network communication.

The .NET framework allows execution of applications written in many different languages, by use of a Common Language Infrastructure (CLI). The source code is first compiled into platform-neutral language called Common Intermediate Language (CIL) by an intermediary compiler. The CIL file is

then compiled by a platform-specific Common Language Runtime (CLR) into bytecode.

The CLR provides the appearance of a virtual machine, so that programmers do not need to take the capabilities of the specific CPU type into consideration. The CLR also provides other important services such as security mechanisms, memory management, and exception handling. The class library and the CLR together compose the .NET Framework.

5.2.2 Visual Basic .NET

As mentioned above, programs written in any language that has a .NET compiler can utilize the .NET framework. The most common languages used with .NET are Microsoft's Visual Basic .NET and Visual C#. However, since all .NET compatible languages use the .NET set of classes for their functionality, the difference between them is merely syntactical.

The choice of language for the implementation fell on Visual Basic .NET mainly because it was the language I personally had the most experience of. Although I had only coded in Visual Basic version 6 (VB6) before, and there are quite a lot of changes in the semantics from this version, I found that getting into Visual Basic .NET was not that big of a struggle.

5.2.3 Design tools

The design and development tool selected for the implementation of the Topology Analysis Engine was Microsoft Visual Basic 2005 Express Edition. This is a free development application offered by Microsoft for smaller projects which do not need the functionality that the more advanced Visual Studio .NET provides. For a screenshot of the development environment see Figure A.3 in the Appendix.

5.3 Implementation

The Discovery runner is implemented as a solution in Microsoft Visual Basic .NET with the main functions in a dynamic linked library (DLL), available to the rest of the system as a COM+ component, and an executable file which calls these functions. In Figure 5.1 the process of the Topology Analysis Engine is shown in the form of a flowchart. This process is described in more detail below.

5.3.1 Initialization

The Topology Analysis process is initiated by retrieving the information from the NETadmin central database about which networks are configured for automatic discovery and which Discovery Agents are responsible for each of these networks.

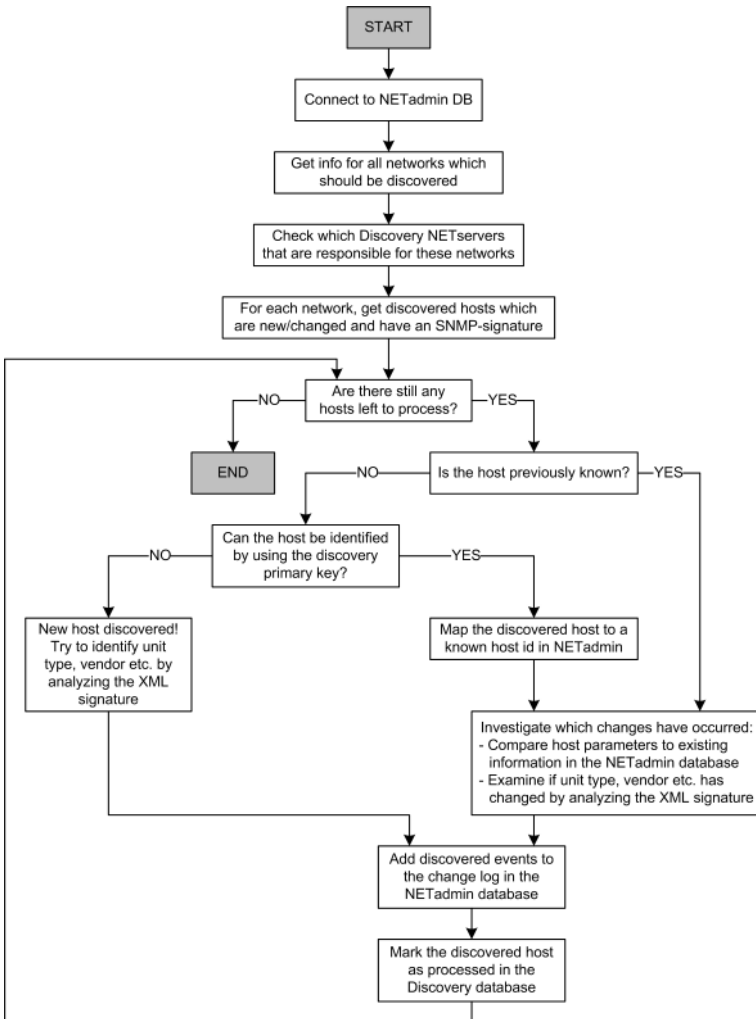


Figure 5.1: Flowchart of the Topology Analysis Engine

5.3.2 Host processing

Each Discovery Agent is contacted in turn, to collect the information about new hosts which have been discovered, or hosts which have had configuration changes. A change is marked by a certain status flag in the Discovery Agent database, so hosts which have been previously processed can be skipped.

When the information about a specific host has been transferred, it is first analyzed to decide if it is a previously known host, i.e. if there exists a mapping between the host and an entry in the NETadmin off-line database.

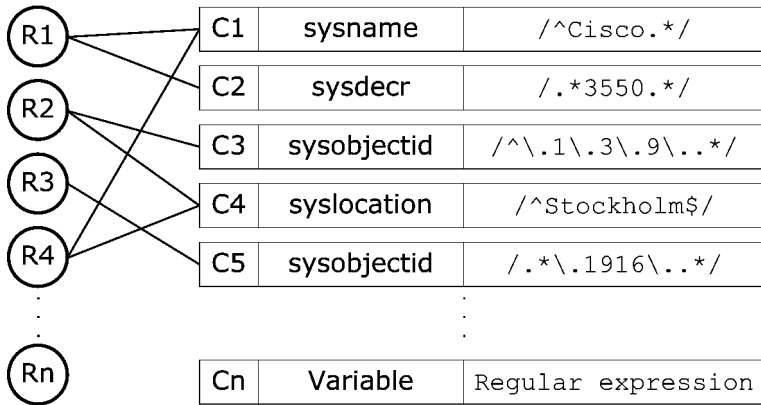


Figure 5.2: An example of rules and conditions used for host classification

5.3.3 Signature analysis

If a host is previously known by NETadmin, a further analysis of the host is done. The host XML signature is parsed and analyzed in order to determine what change has occurred. Different types of changes may be configured by the system administrator, depending on what information is included in the XML signature. In the current implementation the available host characteristics are limited to the ones displayed in Table 4.1. Any found discrepancies are added to the change log which is presented via the User Interface, see Figure A.7 in the Appendix.

5.3.4 Host identification

When a new host is discovered which is not previously known, the engine tries to match it against the discovery primary key. This is done by searching for entries matching the hosts IP address, MAC address, system name, or any other characteristic which has been set as uniquely identifying by the system administrator. If the host can be identified, it is mapped to a host ID in the off-line database, and is then further processed via the XML parsing process described in the previous step.

5.3.5 Classifying unknowns

If a new, unknown host has been discovered, and it cannot be mapped to any equipment in the off-line NETadmin database by host identification, a secondary analysis process is commenced with the purpose of classifying the host as far as possible according to a defined set of rules, see Figure 5.2. Each rule points to an attribute, such as a unit type, a specific vendor, a geographic location etc., which might be valuable for the system administra-

tor in order to understand what kind of new host that has been discovered and if it is legitimate. Each rule consists of a number of conditions, formulated as a variable and a regular expression [19]. If each of the conditions corresponding to a certain rule evaluate to “true”, the rule is matched.

As an example, consider rule R1 in Figure 5.2. It consists of the condition tuples C1:{`sysname` , /`^Cisco.*`/} and C2:{`sysdescr` , /`.*3550.*`/}. Rule R1 will be matched if conditions C1 and C2 both evaluate to “true”, which will occur if the variable `sysname` is a string which begins with “Cisco” and the variable `sysdescr` is a string which contains the sequence “3550”. The rule can then point to an equipment type in the NETadmin database, and the system administrator may be presented with the conclusion that, in this case, a Cisco Catalyst 3550 series switch has been discovered. The more rules that are matched, the more detailed information may be presented regarding the discovered node.

5.3.6 Cycle completion

When all the new or changed hosts from all the Discovery Agents have been analyzed the process is ended. A new analysis process is started when the runner is re-activated by the Windows scheduler.

Chapter 6

User Interface

6.1 Design

All user configuration of the NETadmin system is done through the same graphical user interface (GUI) as the one used to access the normal functions, such as network monitoring, case handling and end-user management. This interface has also been used for the configuration of the discovery system, through an extension of the network management section.

6.1.1 Languages

Administration and configuration of the discovery system is done through an extension of the existing NETadmin graphical user interface. This is a web-based environment, with Microsoft Internet Information Services (IIS) serving Active Server Pages (ASP) over secure HTTP (HTTPS). The development of the ASP code was performed in Microsoft Visual InterDev, shown in Figure A.4 in the Appendix.

6.2 Implementation

In Figure A.5 a screenshot of the NETadmin GUI is presented. It shows the network browser, where networks can be added, removed or configured. For the discovery service implementation, two additions have been made to this section.

6.2.1 General settings

First a radio button control has been added to the configuration template for the general networks settings of an individual network. This is located in the settings area, just below the broadcast setting. By default discovery is disabled for all networks, but may be enabled by this control.

6.2.2 Detailed configuration

The second addition is the detailed discovery settings, which can be reached by the button in the menu bar in the top of the settings area. This is normally disabled (grayed out) when discovery is disabled for a network. From the detailed discovery settings view, shown in Figure A.6, the user can configure the discovery service for the current network. The available settings are:

- **Net scan period**, controls the length of the scan cycle of the network.
- **Host scan period**, the time between scans of individual hosts in the network. If this is set to 0 the scan of the hosts in the network is distributed evenly over the net scan period.
- **Connections scan period**, the time between the scans of links between hosts in the network. Note that this feature is not implemented in the current version of the discovery system.
- **Host key type**, selects which characteristic is considered to be uniquely identifying for a host in the network. See section 5.3.4 for more information about how this property is used.
- **Discovery server**, decides which Discovery Agent server is used for this network.
- **SNMP community list**, contains a list of SNMP communities to try when connecting to discovered hosts in this network. The Auto Fill feature automatically imports known communities from the off-line database.

6.2.3 Storing changes

When configuration of the discovery service has been completed for a certain network, the new settings may be stored by clicking the “Save” button in the menu bar. When this is done, the values are first sanity-checked, and if they are acceptable the new configuration is transferred to the selected Discovery Agent. If there are no problems during this transfer, the user is informed that the new configuration was successfully stored, otherwise an error message is shown.

6.2.4 The discovery log

Another addition to the NETadmin user interface is the network discovery log, illustrated in Figure A.7. This shows the discovered discrepancies in all networks for which the discovery service has been enabled, and is the place where the main output from the discovery and topology analysis services

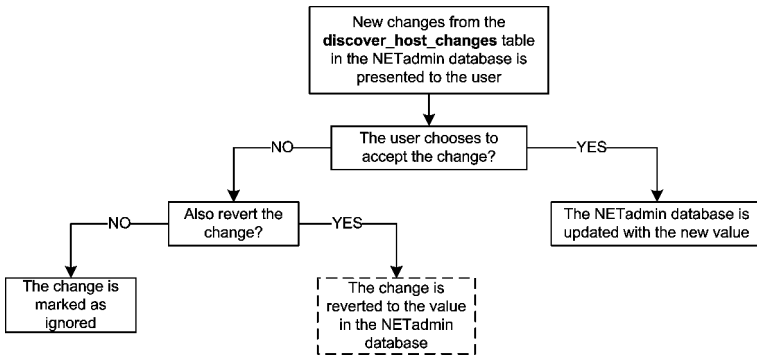


Figure 6.1: Flowchart of the User Interface

are shown. The options available to the user is shown in the flowchart in Figure 6.1.

From this view the system administrator can see all detected new or updated hosts, and choose whether to accept the change or addition of the new host, to reject the change, or to ignore it. If a change is rejected there should also be an option to revert the change, meaning that the changed information (such as a name change of a network host) should be changed back to the value stored in the NETAdmin off-line database. This function is however not yet implemented.

6.2.5 Access control

The NETAdmin GUI contains an access control subsystem, which handles user authentication. It is based on a tiered system, where the network owner, service providers and customer service have separate views. This feature is also used in the discovery service configuration. If there are several different parties using the same NETAdmin interface, only the users responsible for certain networks have access to viewing the discovered changes and new hosts in those networks, or making any configuration changes.

Chapter 7

Testing and evaluation

The complete implementation of the discovery and topology analysis system has been tested in two ways, in an internal test bed system at Netadmin (the staging environment), as well as in a live production system at a customer site. An evaluation of the performance of the system has been carried out in a controlled environment for measuring the overhead and average response times.

7.1 Test environments

A live testing of the discovery system was performed in a real-world environment at a customer company. The company is a regional energy company active in the southwest of Sweden, and operates a metropolitan area network consisting of about 400 active core nodes.

7.1.1 Staging environment tests

Prior to this test, extensive preliminary tests were performed in the so-called staging environment; a copy of the live system, on which new functionality can be installed and tested to verify that it does not conflict with existing components. If any problems arise they can be dealt with, and the system may be rebooted or even completely reinstalled, without any end-users being affected.

These environments are hosted on servers at Netadmin Systems, and are realized by VMware virtualization software. They do not have any contact with the outside world, but are periodically synchronized with the live systems to keep them up to date with settings and end-user data. The tests were used to ensure that no database connections were accidentally left open and that there was no evidence of memory leaks.

7.1.2 Live tests

For the production system tests, two networks at the same company were selected, containing a total of 190 known hosts. The scanning of the networks was successful and the discovery system was able to identify all known hosts in the two networks. A total of 226 discrepancies were found. 79 of these were conflicting system names, where a slightly different naming convention had been used on the physical hosts compared to the one used in the NETadmin system. 143 were conflicting MAC addresses, mainly due to missing MAC address entries in NETadmin system for a majority of the hosts.

On top of this, four new, previously undocumented hosts were discovered. These could be identified by the Topology Analysis Engine, by rules matching the object identifier string (sysObjectID) in the XML signature, as four battery backup units (UPSs). They had apparently been installed by technicians a few months earlier, but had never been properly documented in the NETadmin system, which had led to them subsequently being forgotten. They could however now easily be identified and synchronized into the off-line network model.

7.2 Performance evaluation

Preliminary experiments on the discovery service showed that the time required for one scan of a certain network varied greatly depending on the chosen approach, as shown below. There was also a substantial difference in utilization of system resources depending on the degree of parallelization. Different methods were evaluated to determine the key criteria for scalability, which were identified as CPU resources and database connections. Bandwidth usage by the discovery scans was found to be negligible in all of the evaluated scanning methods.

7.2.1 Top-down implementation

To begin with, a simple top-down solution was tested as a baseline. This method processed all discovered active nodes in the network one by one. Obviously, this was painfully slow and would not scale well with the gradual increase in size of the networks.

7.2.2 Parallelized implementation

Next, the Discovery Agent was divided into smaller parts which could be executed simultaneously. Three distinct activities were identified which could be separated into parallelizable processes. The parallel scanning approach showed a dramatic improvement in scan speed, but also an excessive consumption of system resources; mainly concerning CPU usage and database

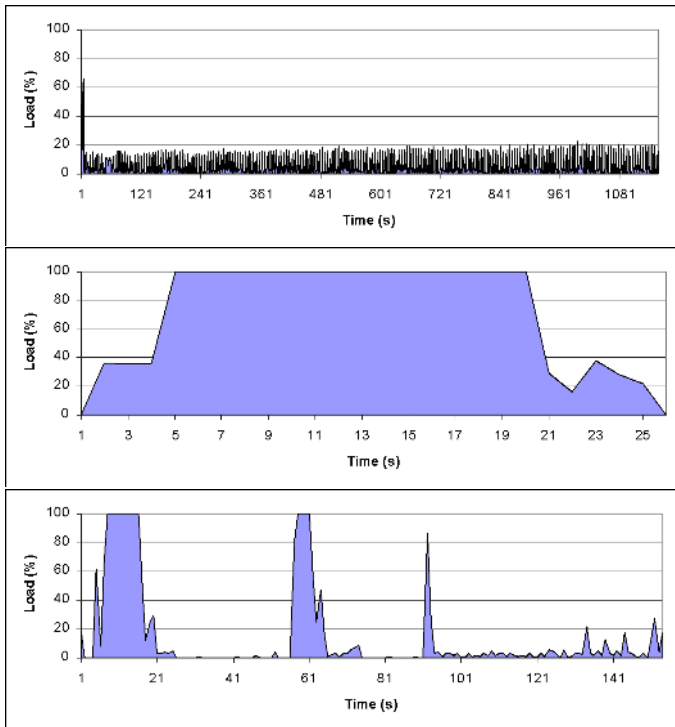


Figure 7.1: CPU load during discovery scans

connections. This became an issue when executing the scripts in a shared environment.

7.2.3 Scheduling evaluation

Finally, a few different scheduling methods were evaluated to control the resource utilization. One way was to implement the scheduling as a function in the scripts themselves. However, this proved to be quite crude for limiting CPU usage. The best result was achieved by making use of the Linux kernel process scheduling subsystem [20], giving the scan processes a slightly lower priority than normal.

The database connection management was implemented directly in the script, with a user configurable minimum amount of connections, or fraction of the available connections, to be reserved for other applications. In this way the networks could be scanned rapidly while still leaving room for other services to execute in the same environment.

7.2.4 Evaluation environment

In order to evaluate the performance of the different approaches above, a special testing environment was prepared. In this environment the scanning scripts and the statistics gathering tool were the only resource-consuming processes running. The NETserver hardware¹, size of the network and number of active nodes (254) were constant between the three runs.

7.2.5 Results

Figure 7.1 illustrates the CPU load while performing the scans and how long each scan took. In the first case a sequential scan was performed, which took 19 minutes and 40 seconds and generated a fairly even load. The time T_s required to scan a network with n active nodes, using this sequential approach, can be expressed by

$$T_s = \sum_{i=1}^n t_i$$

where t_i is the time to scan the individual node i . The second graph in figure 7.1 shows the parallel scan without scheduling, which took a mere 26 seconds, but consumed all available CPU resources. Given unlimited resources, the time T_p to scan a network using this parallel approach, can be expressed by

$$T_p = t_{max}$$

where t_{max} is the time to scan the node which takes the longest, since all nodes are scanned simultaneously. The bottom graph in figure 7.1 illustrates the effects of the resource scheduler with some specific settings. Peaks of heavy processor usage occur until the limit of maximum allowed database connections is reached, and the process has to wait until new connections are freed. A scan using this method took 2 minutes and 34 seconds with the chosen settings.

¹Intel Core Solo T1350 CPU @ 1.86 GHz, 512 MB RAM

Chapter 8

Discussion

8.1 Results

This thesis has described the design and development of a network discovery and topology analysis service, implemented as an extension to the network management and provisioning system NETadmin. All in all, the project resulted in roughly 2000 lines of source code.

Testing of the complete implementation has shown evidence of accurate discovery and classification of unmatched hosts in both staging environments, and in a live customer production network.

The results of the performance evaluation show that the selected parallel approach, with a resource scheduling mechanism, is significant in order to achieve system scalability. The performed tests and evaluations are not by themselves conclusive, but along with the presented analytical rationale they serve as an indication that the implementation is adequate, that it can handle scalability issues to some degree, and that it provides a working mechanism for regulating system resource consumption in a shared environment.

8.2 Future work

The network discovery and topology analysis service implemented and presented in this thesis is not static, but can be further developed with functions for additional reliability, usability and performance. There is currently ongoing work at Netadmin to extend the system, where the focus lies on work which expands the topological model of the network and increases the degree of rogue node detection.

- By probing network elements for their connections to other nodes, using their respective forwarding information base (FIB) and a variety of different discovery protocols, a more comprehensive model of the network can be generated. This allows for discovery of both nodes and edges in the network graph.
- Supplementing the Discovery Agent with a passive scanning engine makes it possible to detect rogue nodes more efficiently, especially those who do not react to active scanning stimuli.
- Collection of DHCP traffic, by use of techniques such as DHCP snooping and the DHCP Option 82, can give further information about hosts entering the network.
- Increasing the support for vendor-specific hardware drivers improves the possibility to better identify certain network equipment and to collect detailed data and statistics.

8.3 Conclusions

The presented architecture, and its incorporated methods, has shown to offer an effective service for flexible network node discovery and topology analysis. Implemented as an add-on module to an existing network management system, it has demonstrated the benefit of synchronization between off-line data and real-world network topology. Using XML and regular expressions to define and evaluate parameters and conditions makes the system highly adaptable and scalable for future improvements.

This work thus lays the ground for further extensions in a security-aware enterprise network management and monitoring system. Adding further data, e.g. about current system versions or patch levels, is a straightforward process and prepares for automatic vulnerability assessment of the nodes in an enterprise network. Future work includes the combination of such a service with a risk analysis and threat assessment infrastructure for business-critical systems.

Glossary

ASP

Active Server Pages. A Microsoft technology for server-side script processing, commonly used for creating dynamic web content. ASP scripts are usually developed in VBScript language.

C++

A general-purpose programming language with both high-level and low-level capabilities. It is generally considered to be an efficient language, known to not incur any overhead for features which are not used.

C#

An object-orientated language developed by Microsoft for the .NET platform, syntactically close to C++ and Java. It can be used both as a compiled language on a computer or as a .NET language.

CIL

Common Intermediate Language. The lowest-level human-readable programming language in the CLI. Languages which target the .NET Framework compile to CIL, which is assembled into bytecode that can be executed by the CLR virtual machine.

CLI

Common Language Infrastructure. An open specification developed by Microsoft that describes the executable code and runtime environment that form the core of the Microsoft .NET Framework.

CLR

Common Language Runtime. The virtual machine component of Microsoft's .NET initiative. Allows programmers to ignore many details of the specific CPU that will execute the program, and provides services like memory management and security features.

COM+

Component Object Model. A Microsoft interprocess communications technology, used to let programs exposes their functionality to other programs through one or more interfaces.

CPU

Central Processing Unit. The component in a digital computer capable of executing a program.

DHCP

Dynamic Host Configuration Protocol. A protocol used to dynamically assign IP addresses and other network parameters to clients in a network.

DLL

Dynamic-Link Library. The Microsoft implementation of dynamic libraries, which are pre-compiled subroutines which may be loaded into an application program at runtime, rather than being linked in at compile time, and remain as separate files on disk ,

DNS

Domain Name System. A system used in IP networks to translate between human-readable computer hostnames and their associated IP addresses.

FIB

Forwarding Information Base. A table in a bridge or a router, containing information about which addresses can be found on which interface.

HTTP

Hypertext Transfer Protocol. A communications protocol used to transfer or convey information on the World Wide Web.

HTTPS

Secure Hypertext Transfer Protocol. Scheme used for accessing resources via HTTP in a secure fashion. It combines the use of the HTTP protocol over an encrypted Secure Sockets Layer (SSL) or Transport Layer Security (TLS) connection.

ICMP

Internet Control Message Protocol. A protocol used in IP networks to transmit error messages and for diagnostic purposes.

IETF

The Internet Engineering Task Force. An organization which coordinates and promotes Internet standards.

IIS

Internet Information Services. Internet-based services for servers using Microsoft Windows. It is the world's second most popular web server in terms of overall websites, behind the Apache HTTP Server.

IP address

Internet Protocol address. The address used in the network layer to uniquely identify a host communicating via the Internet Protocol.

IPTV

Internet Protocol Television. A system where digital television content is delivered by using the Internet Protocol, over a network infrastructure, instead of being distributed through traditional broadcast or cable formats.

Java

An object-oriented programming language originally developed by Sun Microsystems. Java applications are typically compiled to bytecode which can run on any Java virtual machine (JVM) regardless of computer architecture.

LLDP

Link Layer Discovery Protocol. A vendor-neutral data link layer protocol, which allows a network device to advertise its identity and capabilities on the local network. Formally ratified as IEEE standard 802.1AB in May 2005.

MAC address

Media Access Control address. The address used in the data link layer to uniquely identify a network adapter in the local area network.

MIB

Management Information Base. A type of virtual database, comprising a collection of objects, used to manage the devices in a communications network.

MySQL

A multithreaded, multi-user SQL database management system developed by the Swedish company MySQL AB as free software primarily under the GNU General Public License.

OAN

Open Access Network. A network which implements a horizontal layered architecture and business model which separates physical access to the network from service provisioning. The same OAN may be used by a number of different service providers, who either share investment and maintenance costs, or who pay a fee to a network infrastructure operator for end-user access.

OID

Object Identifier. An identifier used to address a node in a hierarchically-assigned namespace.

Option 82

A feature used by multilayer switches, to append information to a DHCP request, regarding which physical port the client is attached to who issued the request.

OSI model

Open Systems Interconnection Basic Reference Model. A layered, abstract description for communications and computer network protocol design.

Perl

A classic UNIX scripting language often used to parse or manipulate text, or similar tasks. It has been extended over the years with support for multiple programming paradigms, and a large collection of third-party modules.

PHP

A popular, free software scripting language mainly used for web applications and handling of dynamic content. PHP's principal focus is server-side scripting, and it contains extensive built-in functionality for database access, text processing, and file handling.

Ruby

An interpreted, object-oriented programming language scripting language with has Perl-inspired syntax. It has become popular for developing web applications, but have been struggling with some performance issues.

SNMP

Simple Network Management Protocol. A protocol commonly used in network management systems to monitor network-attached devices for conditions that warrant administrative attention.

SQL

Structured Query Language. A standard interactive and programming language for retrieving and manipulating information stored in a database.

TCP

Transmission Control Protocol. A transport layer protocol which is used in IP networks. It provides reliable, in-order delivery of a stream of bytes, making it suitable for applications like file transfer and e-mail.

UDP

User Datagram Protocol. A transport layer protocol which is used in IP networks. A fast and efficient protocol which, in comparison with TCP, does not provide any services for reliability. It is commonly used by streaming media applications such as IPTV, Internet radio or online games.

UPS

Uninterruptible Power Supply. A hardware device which maintains a continuous supply of electric power to connected equipment, by supplying power from a separate battery source when utility power is not available. May be attached to a computer network for monitoring and configuration purposes.

Virtual machine

A software implementation of a computer, which executes programs like a real machine.

Visual Basic .NET

An object-oriented computer language which is an evolution of Microsoft's Visual Basic, with the purpose of being used with the .NET framework.

Visual InterDev

An integrated development environment which is a part of Microsoft Visual Studio. It is mainly used for creating ASP applications, and provides access to code completion functionality, database server management tools, and an integrated debugger.

VMware

A brand of proprietary virtualization software for x86-compatible computers.

XML

eXtensible Markup Language. A general-purpose, free, open standard markup language, used to facilitate the sharing of structured data across different information systems.

Appendix A

Screenshots

This appendix contains illustrating screenshots from the developing environments and the NETadmin system user interface.

Figure	Description
A.1	PHPEclipse developing environment. This screenshot shows the source code of a module in the Discovery Agent being developed in PHPEclipse.
A.2	MySQL Control Center. This application is used to monitor the contents of the discovery database to verify that the correct values have been received for each host.
A.3	Visual Basic .NET environment. Microsoft Visual Basic 2005 was used to develop the .NET application for the Topology Analysis Engine.
A.4	Visual InterDev. ASP script code was used to create the graphical system user administration interface.
A.5	General network settings. In this view the configuration settings for a specific network is shown. The Discovery service can be enabled by toggling a radio button.
A.6	Detailed discovery settings. Each network can be configured with separate specific settings for the discovery service, such as which Discovery Agent to use, and how often to scan.
A.7	Network discovery log. Information about changes in the network is presented in the discovery log. In this view the system administrator can choose what action to take for each change.

Table A.1: Description of the figures in this appendix

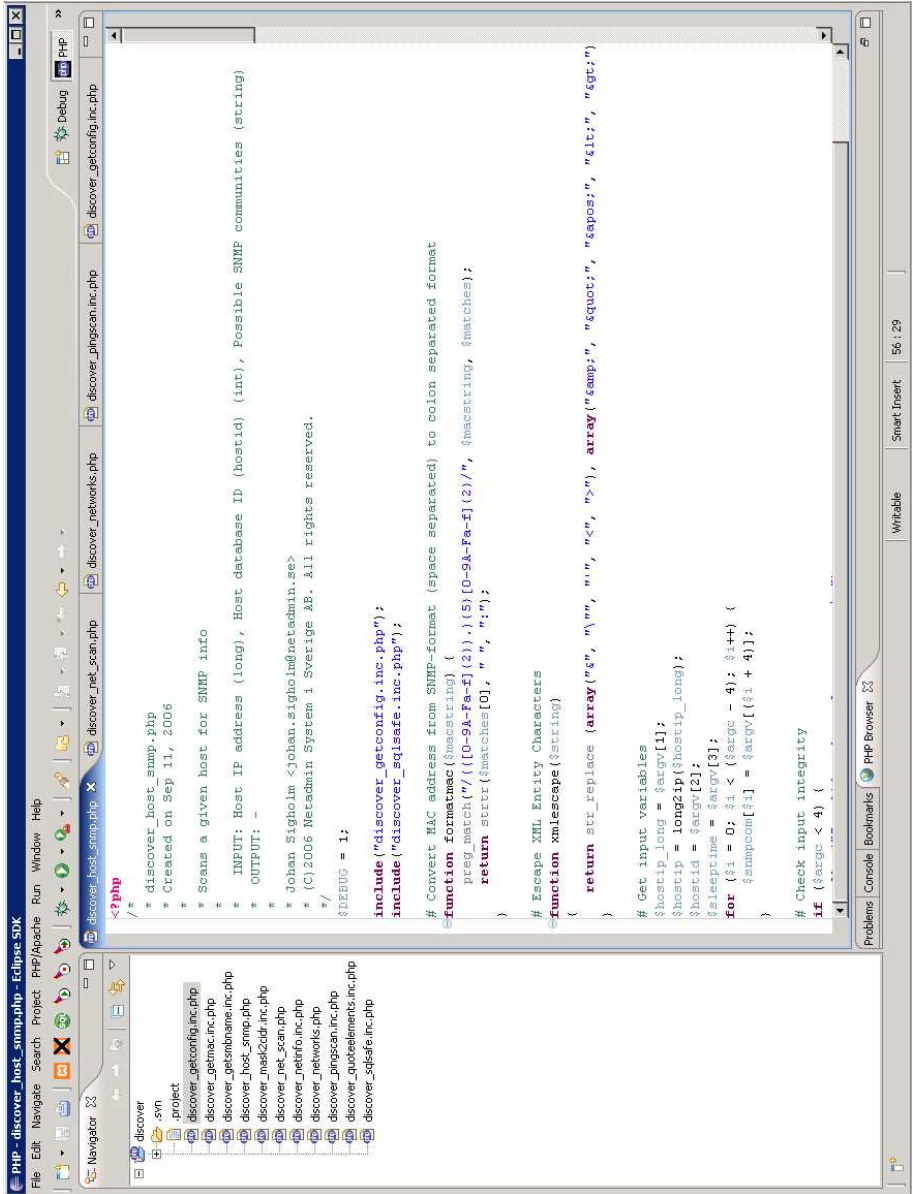


Figure A.1: PHPEclipse developing environment

	hostid	hostaddress	hostnameid	hostcommunity	hostid	hostmanufactured	hostmac	hostsignature	hostshardid	hostname
1	184418305		1		0	00E04C42A81C3		2065208102650		
2	184418306		1		0	000130662F80	<?xml version="1.0" encoding="utf-8" ?>	2065211135207		
3	184418307		1	public	0	0001033618D00	<?xml version="1.0" encoding="utf-8" ?>	2065208102650		
4	184418308		1		0	00E08121F13A		2065208102650		
5	184418309		1		0	00E04C42A81C3		2065208102650		
6	184418306		1		0	000130662F80		2065208102650		
7	184418307		1	public	34	0001033618D00	<?xml version="1.0" encoding="utf-8" ?>	2065211135212		
8	184418309		1		36	00E08121F13A		2065208102650		
9	184418361		1		0	00013723C1C94		2065208102650		
10	184418363		1		25	000142278F2F2		2065208102650		
11	184418364		1		0	000137238E5C8		2065208102650		
12	184418365		1		0	00013723C0E81		2065208102650		
13	184418366		1		0	0001372FC24C4		2065208102650		
14	184418367		1		0	0000C29286803		2065208102650		
15	184418368		1		0	0001422D09FC22		2065208102650		
16	184418369		1		0	0001422D09F980		2065208102650		
17	184418370		1		0	000123FCFCFE0A		2065208102650		
18	184418371		1		0	00015C5E1F5D9		2065208102650		
19	184418372		1		0	00015C5FA42395		2065208102650		
20	184418373		1		0	000809F3640F4		2065208102650		
21	184418374		1		0	000809F361466		2065208102650		
22	184419073		1		0	000809F568931A		2065208102650		
23	184419074		1		0	000809F568937		2065208102650		
24	184419075		1		0	000809F5689343		2065208102650		
25	184419076		1		0	000809F5689375		2065208102650		
26	184419077		1		0	000809F5689333		2065208102650		
27	184419078		1		0	000809F56E5D0		2065208102650		
28	184419079		1		0	000809F563488		2065208102650		
29	184419080		1		0	000809F5634CE		2065208102650		
30	184419081		1		0	000809F568932E		2065208102650		
31	184419082		1		0	000809F568933A		2065208102650		
32	184419084		1		0	000809F563422		2065208102650		
33	184419085		1		0	000809F563425		2065208102650		
34	184419086		1		0	000809F56E0E		2065208102650		
35	184419087		1		0	000809F56E0E5		2065208102650		
36	184419089		1		0	000809F563421		2065208102650		
37	184419331		1		0	00015C515A24C		2065208102650		
38	184419333		1		0	000123FC0F04C2		2065208102650		
39	184419335		1		0	000123FC012636		2065208102650		

Figure A.2: MySQL Control Center

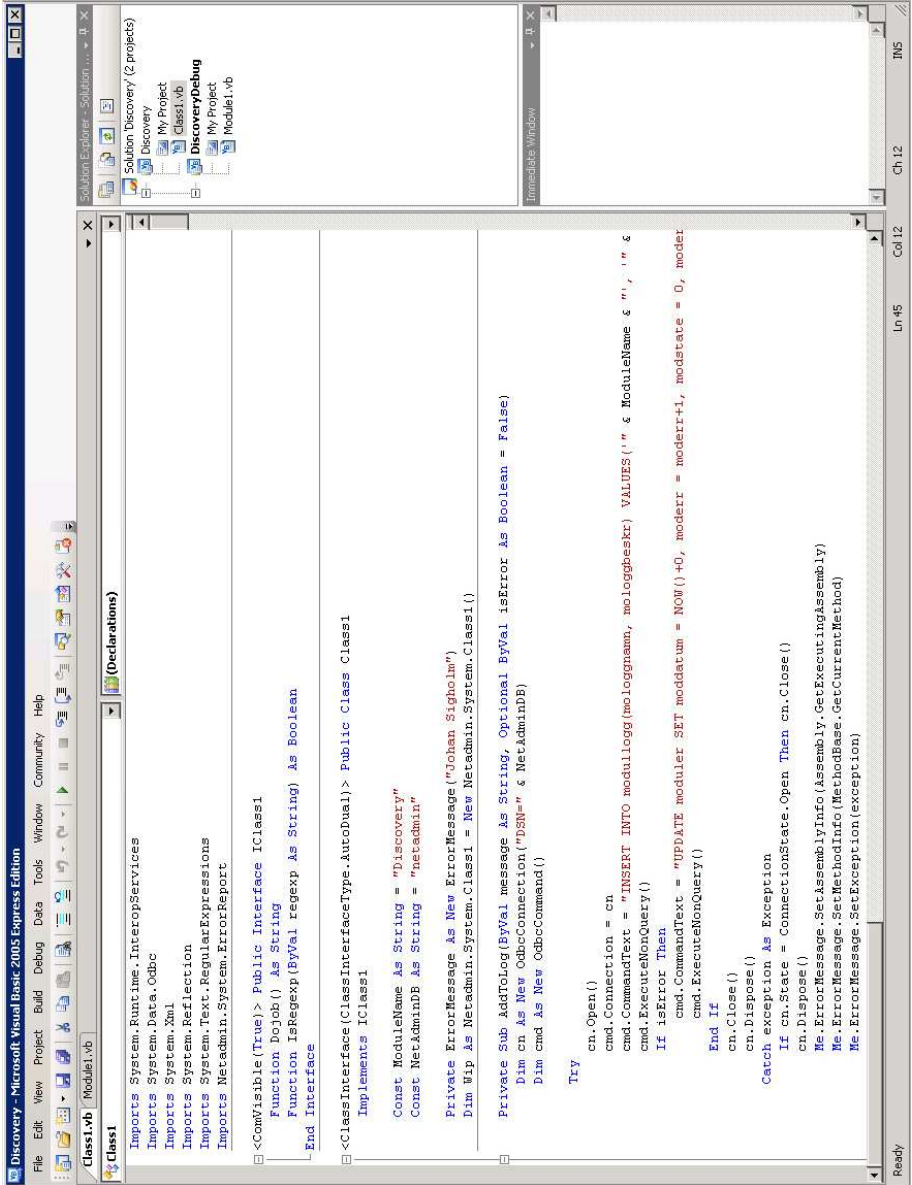


Figure A.3: Visual Basic .NET environment

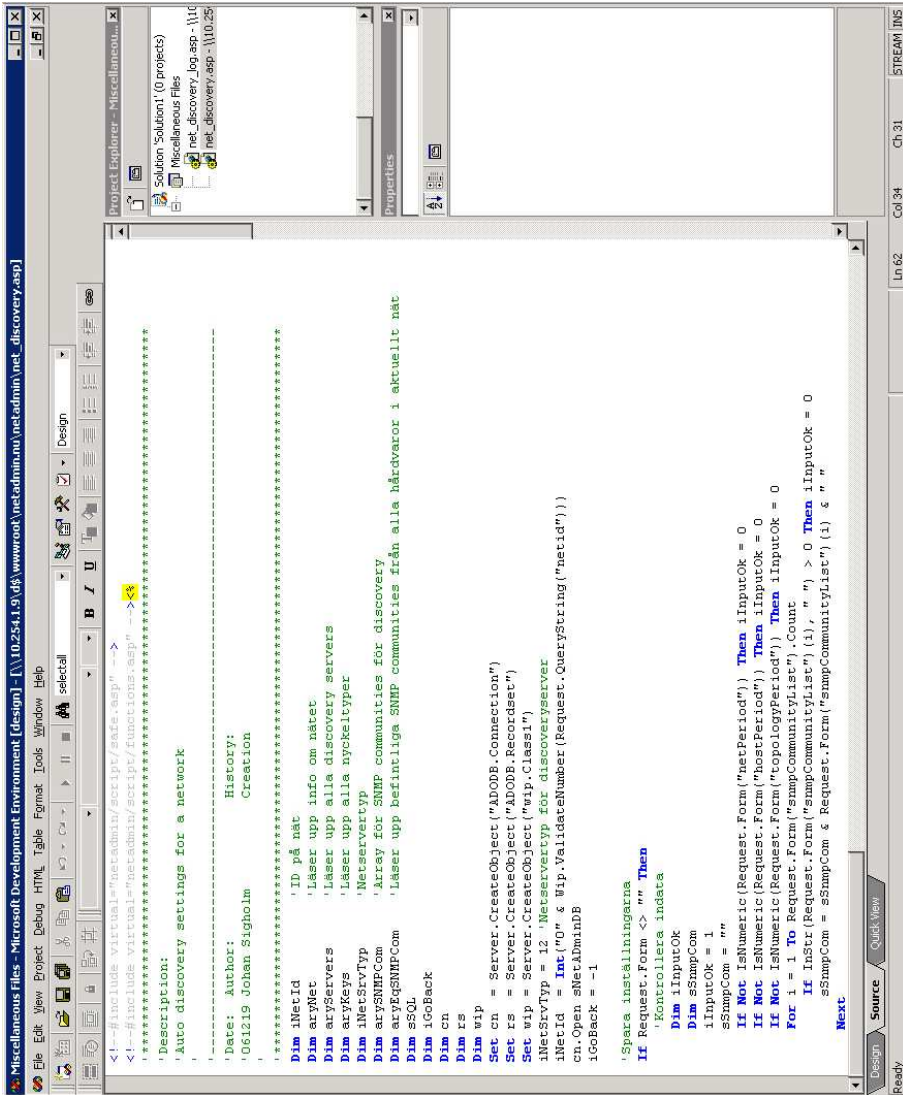


Figure A.4: Visual InterDev environment

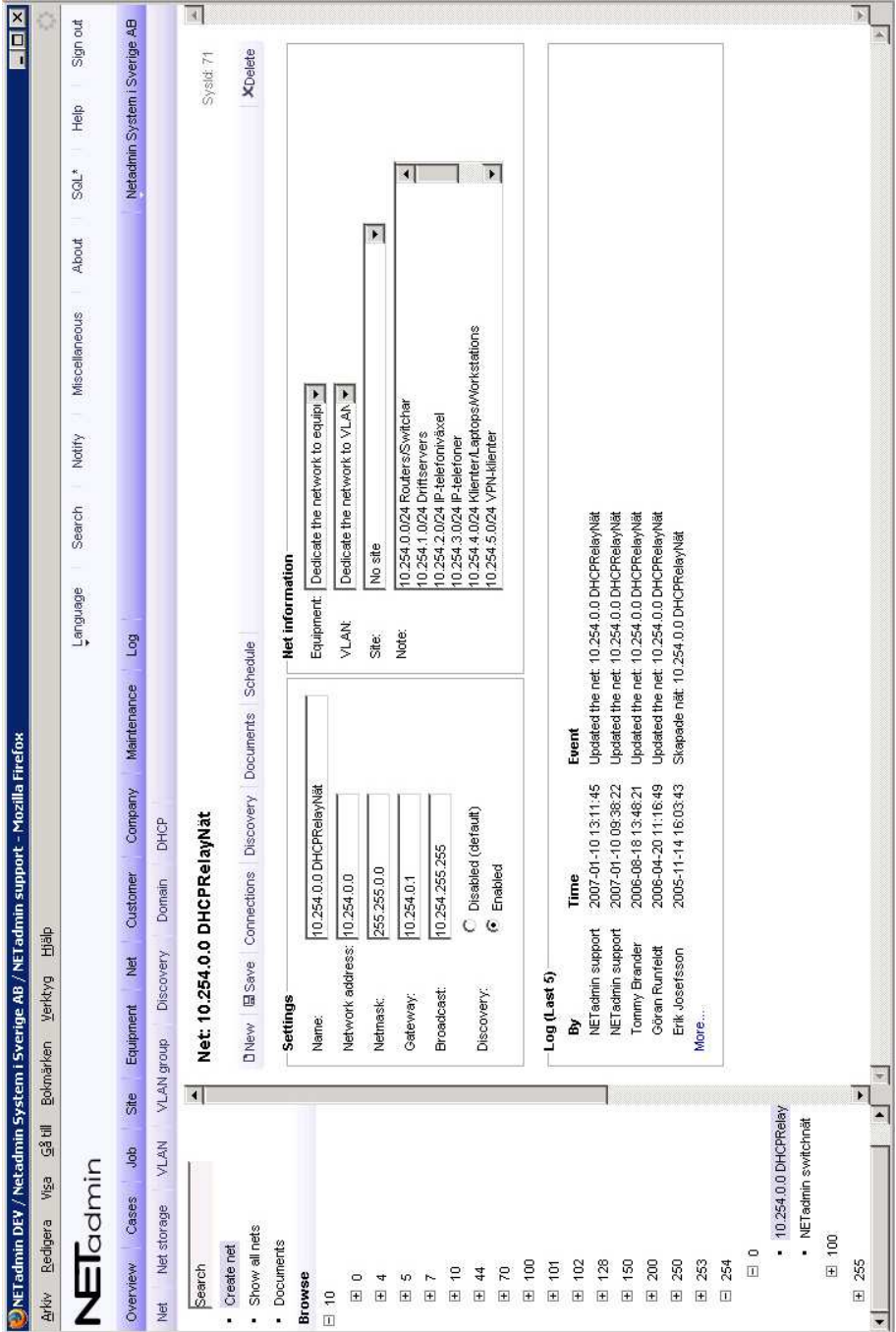


Figure A.5: General network settings

The screenshot displays the NetAdmin web interface in a Mozilla Firefox browser. The page title is "NETadmin DEY / Netadmin System i Sverige AB / NETadmin support - Mozilla Firefox". The navigation menu includes: Arkiv, Redigera, Visa, Gå till, Böckmärken, Verktyg, Hjälp, Language, Search, Notify, Miscellaneous, About, SQL+, Help, Sign out, and Netadmin System i Sverige AB.

The main content area is titled "Discovery: 10.254.0.0 DHCP RelayNät" and includes a "Back" button and a "Save" button. The "Settings" section contains the following fields:

- Net scan period: 60
- Host scan period: 0
- Connections scan period: 0
- Host key type: staging-stat-1
- Discovery server: public
- SNMP community list: secretcommunity, readonly

Below the settings are "Auto fill", "Remove", and "Add" buttons. A "Browse" section shows a tree view of IP addresses from 10 to 255, with "10.254.0.0 DHCP Relay" and "NETadmin switchnät" expanded.

Figure A.6: Detailed discovery settings

NETADMIN / NETADMIN SYSTEMS1 SVRGE AB / NETADMIN SUPPORT - Mozilla Firefox

Overview Cases Job Site Equipment Net Customer Company Maintenance Log

Net Net storage VLAN VLAN group Discovery Domain DHCP

• Documents

Auto discovery log

Discovered changes

Import selected Ignore selected

Select: All, None, New, Changed, Dupes, Non-dupes

<input type="checkbox"/>	Date and time	Event	Information	Number
<input type="checkbox"/>	2006-12-08 10:26:02	Unknown new host discovered	10.254.0.1	1
<input type="checkbox"/>	2006-12-08 10:26:02	Unknown new host discovered	10.254.0.2	1
<input checked="" type="checkbox"/>	2006-12-08 10:26:02	MAC address change	Host netadmin-pl-1 changed to 00E08121F13A	1
<input type="checkbox"/>	2006-12-08 10:26:02	Unknown new host discovered	10.254.1.1	1
<input checked="" type="checkbox"/>	2006-12-08 10:26:02	MAC address change	Host glasa-3 changed to 00F432276B3F2F	1
<input type="checkbox"/>	2006-12-08 10:26:02	Unknown new host discovered	10.254.1.4	1
<input type="checkbox"/>	2006-12-08 10:26:02	Unknown new host discovered	10.254.1.5	1
<input type="checkbox"/>	2006-12-08 10:26:04	Unknown new host discovered	10.254.3.1	1
<input type="checkbox"/>	2006-12-08 10:26:04	Unknown new host discovered	10.254.2.3	1
<input type="checkbox"/>	2006-12-08 10:26:05	Unknown new host discovered	10.254.4.3	1
<input type="checkbox"/>	2006-12-08 10:26:07	Unknown new host discovered	10.254.5.254	1
<input type="checkbox"/>	2006-12-08 10:26:07	Unknown new host discovered	10.254.6.1	1
<input checked="" type="checkbox"/>	2006-12-08 10:26:08	MAC address change	Host terminal-2 changed to 00123F2A4B6A3	1
<input type="checkbox"/>	2006-12-08 10:26:08	Unknown new host discovered	10.254.6.4	1
<input type="checkbox"/>	2006-12-08 10:26:08	Unknown new host discovered	10.254.8.3	1
<input checked="" type="checkbox"/>	2006-12-11 13:52:12	Equipment name change	Host netadmin-switch-2 changed from netadmin-switch-2 to DEFAULT_COMING	1
<input checked="" type="checkbox"/>	2006-12-11 13:52:12	MAC address change	Host netadmin-switch-2 changed to 001083281BD80	1
<input checked="" type="checkbox"/>	2006-12-11 13:52:12	IP address change	Host netadmin-switch-2 changed to 10.254.0.3	1
<input type="checkbox"/>	2006-12-11 13:56:55	Unknown new host discovered	Host netadmin-ssid-4 changed to 10.254.6.10	1
<input type="checkbox"/>	2006-12-11 13:56:59	Unknown new host discovered	Host BELLYBEET changed to 10.254.7.1	1
<input type="checkbox"/>	2006-12-11 13:59:25	Unknown new host discovered	Host BELLYBEET changed to 10.254.7.3	1
<input type="checkbox"/>	2006-12-11 13:59:26	Unknown new host discovered	Host printer-4 changed to 10.254.7.4	1
<input type="checkbox"/>	2006-12-11 13:59:27	Unknown new host discovered	Host ET008-00797418 changed to 10.254.7.5	1
<input type="checkbox"/>	2006-12-11 13:59:30	Unknown new host discovered	Host ET008-00859507 changed to 10.254.7.6	1
<input type="checkbox"/>	2006-12-11 13:59:32	Unknown new host discovered	Host ET008-00798186 changed to 10.254.7.7	1
<input type="checkbox"/>	2006-12-11 13:59:34	Unknown new host discovered	Host printer changed to 10.254.7.8	1
<input type="checkbox"/>	2006-12-11 14:01:00	Unknown new host discovered	Host BELLYBEET changed to 10.254.7.2	1
<input type="checkbox"/>	2006-12-11 14:11:21	Unknown new host discovered	Host glasa-3 changed to 10.254.106.254	1

Select: All, None, New, Changed, Dupes, Non-dupes

Import selected Ignore selected

Figure A.7: Network discovery log

Bibliography

- [1] Battiti R., Lo Cigno R. A., Orava F., Pehrson B., “Global growth of open access networks: from warchalking and connection sharing to sustainable business.” In *Proc. 1st ACM WMASH Workshop*. September 2003.
- [2] The National Post and Telecom Agency (PTS), “IT infrastructure for town and country.” SOU 2000:111. ISBN 91-38-21347-8. November 2000.
- [3] Gamez D., Nadjm-Tehrani S., Bigham J., Balducelli C., Chyssler T., Burbeck K., “Safeguarding Critical Infrastructures.” Chapter in H. B. Diab, A.Y. Zomaya (Eds.), *Dependable Computing Systems: Paradigms, Performance Issues and Applications*. John Wiley & Sons, Inc. ISBN 0-471-69461-4. November 2005.
- [4] Han C.C., Kumar R., Shea R., Srivastava M., “Sensor network software update management: a survey.” In *Proc. ACM International Journal on Network Management*. Vol. 15. July 2005.
- [5] Greenberg, A., Hjalmtysson, G., Maltz, D. A., et al., “A clean slate 4D approach to network control and management.” *ACM SIGCOMM Computer Communications Review*. October 2005.
- [6] Barthel A., “Analysis, Implementation and Enhancement of Vendor dependent and independent Layer-2 Network Topology Discovery.” Diploma Thesis, Chemnitz University of Technology, Chemnitz, Germany, April 2005.
- [7] Vigna G., Valeur F., Zhou J., Kemmerer R.A., “Composable Tools For Network Discovery and Security Analysis.” In *Proc. Annual Computer Security Applications Conference (ACSAC)*. December 2002.
- [8] Netdisco. <http://netdisco.org/>.
- [9] Tenable Network Security, Nessus Open Source Vulnerability Scanner Project. <http://www.nessus.org/>.
- [10] HP OpenView. <http://www.openview.hp.com/>.

- [11] Microsoft Systems Management Server.
<http://www.microsoft.com/smsserver/>.
- [12] The Open Systems Interconnection. "Open System Interconnection Basic Reference Model." ISO/IEC 7498-1. 1994.
- [13] PHPEclipse. <http://www.phpeclipse.net/>.
- [14] Shankar U., Paxon V., "Active Mapping: Resisting NIDS Evasion Without Altering Traffic." In *Proc. 2003 IEEE Symposium on Security and Privacy*. May 2003.
- [15] Thomas, Y., Debar, H., Morin, B., "Improving security management through passive network observation." In *Proc. First International Conference on Availability, Reliability and Security, ARES 2006*. April, 2006.
- [16] Fping. <http://fping.sourceforge.net/>.
- [17] Nmap. <http://insecure.org/nmap/>.
- [18] Xprobe2. "Active OS fingerprinting tool."
<http://xprobe.sourceforge.net/>.
- [19] Friedl, J., "Mastering Regular Expressions, 3rd Edition". O'Reilly Media Inc. ISBN 0-596-00289-0. August 2006.
- [20] Bovet D. P., Cesati M., "Understanding the Linux Kernel, 3rd Edition." O'Reilly Media Inc. ISBN 0-596-00565-2. November 2005.

Index

- .NET framework, 25
- Active Server Pages, 31
- ASP, 31
- C++, 17
- Cisco Discovery Protocol, 15
- Digital Maintenance Operation Protocol, 15
- Discovery Agent, 4, 11, 17, 19, 22, 27, 32, 36
- Extreme Discovery Protocol, 15
- HTTP, 31
- HTTPS, 31
- IPTV, 3
- Java, 17
- Link Layer Discovery Protocol, 15
- LLDP, 15
- Management Information Base, 14
- NETadmin, 1, 4, 9
- Netadmin System i Sverige AB, 1
- Object Identifier, 14
- OID, 14
- Open Access Networks, 3
- Perl, 17
- PHP, 17
- primary key, 28
- Ruby, 17
- Runner, 11, 26
- Simple Network Management Protocol, 14
- SNMP, 14
- Topology Analysis Engine, 4, 11, 22, 25, 26
- User Interface, 4, 19, 28, 31
- video on demand, 3
- Visual Basic .NET, 26
- Visual C#, 26
- Visual InterDev, 31
- Wasadata System AB, 2
- XML, 5, 17, 22, 28, 36

