WindowsResearch

Implementation of a Translator for Windows Machines
within the Project VeriNeC

Master Thesis in Informatics
University of Fribourg, Switzerland

Author:
Nadine Zurkinden
Waldweg 6
3186 Düdingen
nadine.zurkinden@unifr.ch

Supervisor:
Prof. Dr. Ulrich Ultes-Nitsche
telecommunications, networks & security
Research Group
Department of Computer Science

Assistants:
David Buchmann
Dominik Jungo

September, 2005

Abstract

This paper shows how Windows machines can be remotely configured using a Java API. It is part of the project Verinec. Verinec aims to simplify network configuration. The idea is to describe the abstract definition of a network using the XML syntax. This abstract configuration will be translated automatically into machine specific configuration using XSLT. Then it will be distributed to the target system using the Windows Management Instrumentation (WMI). Through WMI, Windows resources can be managed locally or remotely over a network using scripts. The Java API used to access the WMI is Jawin.
The configuration of the Ethernet card and dial-up modem was studied in detail. It was also researched, how the user account management is handled in Windows.

Keywords: Network Configuration, Windows Management Instrumentation, Jawin

Contents

List of Figures

Chapter 1

Introduction

Configuring a network of computers is more and more complex as environments become heterogeneous. Also the number of supported services needed to be configured increases.
The project Verified Network Configuration (VeriNeC), founded by the Swiss National Science Foundation, shall simplify network configuration. Therefore computers are configured automatically. Another aim of Verinec is to help configuring secure networks. For this purpose a simulator checks whether the network fulfills the desired behaviour. If the simulated network behaves as desired, the configuration data can be distributed to the target computer.
The automatic configuration of Linux machines is quite simple as the configuration is based on files. So the text files with the configuration data can simply be stored to the appropriate directory in the file system. The configuration of Windows machines on the contrary is mostly based on a registry. Therefore another solution to configure network services must be found.

1.1 Objectives

The aim of this thesis is that Windows machines can be configured automatically through Verinec. To achieve this, the project Verinec must be understood, particularly the translation process. Then a research is made on how Windows can be configured remotely using a Java Application Programming Interface (API). Based on that research, the Windows translator will be implemented. Therefore translation files will be created using the eXtensible Stylesheet Language Transformations (XSLT). Then the distributor will be implemented. It should be universal for all services configurable through WMI, also for those not yet supported but which follow.
The research is made globally and the configuration of the Ethernet card will be implemented. There is also a research on how the user account management is handled in Windows, but this will not be implemented.

1.2 Overview

After this introduction, the paper starts with a short summary of the Verinec project. The architecture of Verinec is shown and the function of the translator is explained briefly.
The third chapter covers the research procedure: How Windows machines can be configured automatically using a Java API. Three approaches with corresponding Java API's are presented. A short overview of the chosen alternative concludes that chapter.
In the fourth chapter, the Windows Management Instrumentation (WMI) is exposed in detail. The WMI architecture is explained on the basis of managed resources, WMI infrastructure and consumers. As the Common Information Model (CIM) is very important in order to work with WMI, it is explained in an own section. Then the functionality of the scripting API for WMI is illustrated. Furthermore it will be shown, how an Ethernet card can be configured through WMI. To finish this chapter, an outlook how the user account management can be handled in Windows using WMI is presented.
The fifth chapter is about Jawin. It shows which preconditions must be considered before one can work with Jawin. There is also an example explained in detail.
In the sixth chapter the translation process of the Windows Translator is described in detail.
The last chapter contains the conclusions of the project.

Chapter 2

VeriNeC

Configuring a network can be very difficult, as there may be different environments in a network. The main idea of the project Verinec is to simplify the process of network configuration.
The basic idea of Verinec is to describe the abstract configuration data using an eXtensible Markup Language (XML) syntax. The abstract configuration is automatically translated into machine specific configuration and distributed to the specified system. Prior to really configure the network, the simulator allows testing whether the configuration fulfills the desired behaviour.

2.1 Architecture of VeriNeC

The architecture of Verinec is made up of several modules. Figure 2.1 shows the different Modules of the Verinec system.

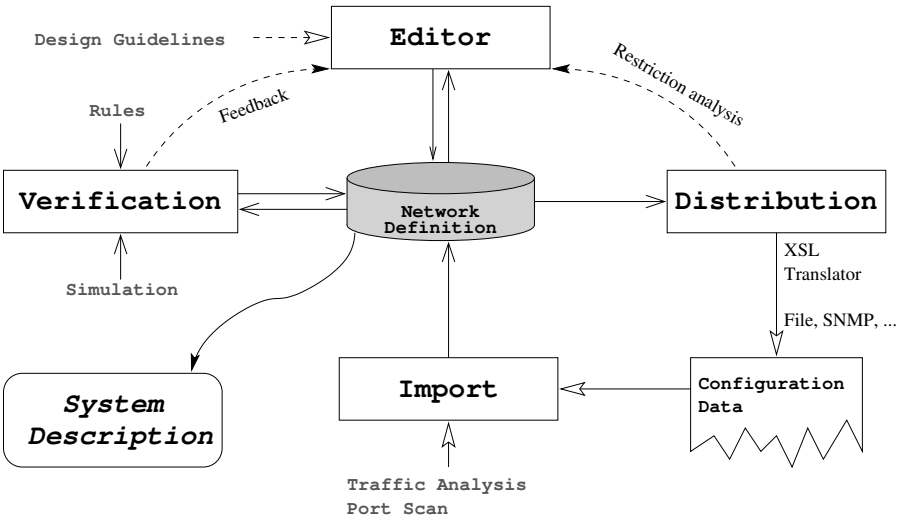

Figure 2.1: Architecture of Verinec [1]

- The **Network Definition** is the base of Verinec and contains all configuration details stored in XML structure.
- The **Verification** module is testing whether a network fulfills specified requirements. A **Simulation** module aims to build a virtual network and simulate the network behaviour.
- The **Editor** is a Graphical User Interface (GUI), which displays a network and helps to generate the network configurations.
- The **Distribution** module translates abstract configuration into system specific configurations and distributes these to the specified system.
- The **Import** module analyzes networks and configuration files to simplify the introduction of the Verinec system in existing environments.
- The **System Description** module is a graphical application that controls the different modules.

2.2 Distribution

The translation module generates configuration files from the abstract XML data. The translation is done in two steps.
First, the abstract configuration data is translated into concrete configuration, which is applicable for the distributor. Using metadata in the configuration document the translator selects the correct translation XSLT and distributor, which is a Java class, for each service. Because of unsupported features in some systems, the translator has a restriction module which can produce warnings to the user. These restrictors extract warnings for configurations that can not be fulfilled for the selected architecture.
The second step of distribution consists of pre- and post processing (i.e. stopping and starting services) and the application of the configurations. The distributor is implemented in Java and puts the appropriate configuration data on the target system.

For more information about Verinec see [1] and [2]. The translator will be explained more detailed in Chapter 6.

Chapter 3

Research

The configuration of Windows machines is mostly based on the registry. Part of this research thesis was to evaluate the different means in which Windows can be remotely configured using a Java API. There are three approaches illustrated below.

3.1 Registry

One obvious possibility is to modify the registry directly. The registry is the basis for the whole configuration in Windows (since Windows 95). It is a hierarchically structured database where all configuration data of the operating system, installed applications and hardware devices is stored.

The hierarchy is the basis for the organization of the registry. The registry is composed of thousands of keys. Every key can have one or more entries as well as child keys. Each entry consists of a name, a data type and a value. The hierarchical structure of the registry is very similar to the file system of the explorer. Figure 3.1 shows how the registry is structured.

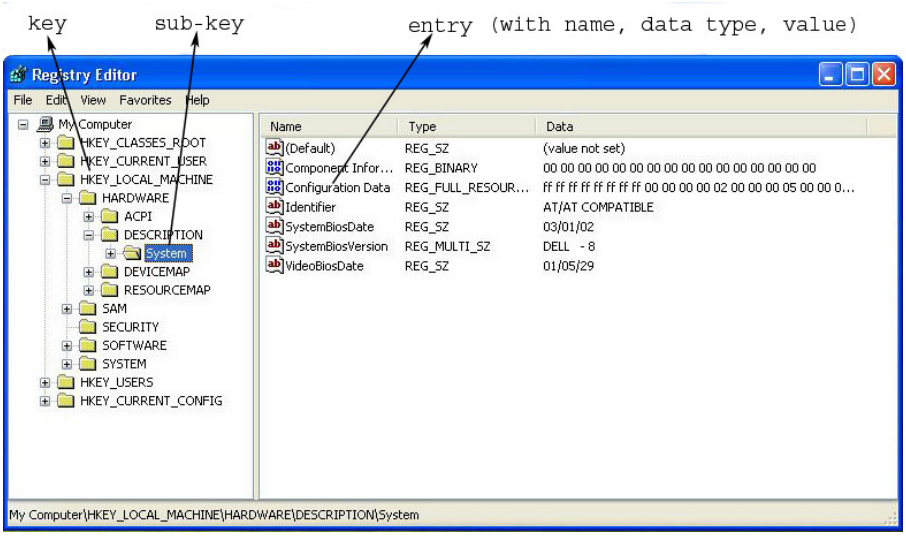

Figure 3.1: Structure of the Windows Registry [3]

In Windows XP the hierarchical structure of the registry begins with five keys which just act as logical classification and contain no entries. These are:

- **HKEY_CLASSES_ROOT**: In this key all information of Component Object Model (COM) objects, file extensions and file type is stored. It is the alias of HKEY_LOCAL_MACHINE/Software/Classes.
- **HKEY_CURRENT_USER**: All software settings of the user currently logged on the machine are achieved here, including desktop settings, logon name and start menu settings.
- **HKEY_LOCAL_MACHINE**: This is the most important key. Here is the configuration data of Windows, the applications and the hardware stored.
- **HKEY_USERS**: The individual preferences of all registered users are saved here. The keys of the actual user are loaded to HKEY_CURRENT_USER.
- **HKEY_CURRENT_CONFIG**: This key contains the data of the actual hardware configuration. It corresponds to the key HKEY_LOCAL_MACHINE/System/CurrentControlSet/HardwareProfiles.

For more details about the Windows registry see [3].

3.1.1 Java API for Registry

Two alternatives were studied about how one can access the Windows registry using Java. Probably there exist other.

jRegistry Key

The first one is the jRegistry Key Java Native Interface (JNI) API [4]. It was designed by BEQ Technologies Inc. to facilitate Windows registry access for Java developers. They have decided to release jRegistry Key as an open-source product under the LGPL. A few classes are provided to manipulate the Windows registry. These classes are listed in Figure 3.2.

Package ca.beq.util.win32.registry

Class Summary	
KeyIterator	`KeyIterator` provides methods to iterate through the subkeys of a particular registry key.
RegistryKey	A representation of system registry keys, `RegistryKey` is the principle class of the `ca.beq.util.win32.registry` package.
RegistryValue	A representation of registry values.
RootKey	The Windows operating system defines standard registry keys ("root keys", represented by `RootKey`) that are always open.
ValueIterator	`ValueIterator` provides methods to iterate through the values of a particular registry key.
ValueType	Registry values (data) can be stored in various formats (types), represented by `ValueType`.

Figure 3.2: Classes of jRegistry Key [4]

An example how the computer name can be changed with jRegistry Key is shown in Figure 3.3.

Example with jRegistry Key

```
import ca.beq.util.win32.registry.*;

public class jRegistryKey {
  public static void main( String[] args ) {
    //the path of the key you will access
    RegistryKey r = new RegistryKey(RootKey.HKEY_LOCAL_MACHINE,
        "System\\CurrentControlSet\\Control\\ComputerName\\ComputerName");
    //ComputerName will be changed to myPC
    RegistryValue v = new RegistryValue("ComputerName", ValueType.REG_SZ, "myPC");
    r.setValue(v);
  }
}
```

Figure 3.3: Changing the Computer Name using jRegistry Key

The example consists of two parts. First, the key which we want to access is specified. Therefore a new `RegistryKey r` with the path of the key will be created. The entry for the computer name is stored under HKEY_LOCAL_MACHINE/System/CurrentControlSet/Control/ComputerName/ComputerName. Then a new `RegistryValue v` will be created with the new parameters name, data type and value. In this example only the value is changed. Finally the new value will be assigned to the key with `r.setValue(v)`.

JNIRegistry

Another possibility is JNIRegistry [5]. It was developed by ICE Engineering Inc. to access, modify and export Windows registry resources with Java. The `com.ice.jni.registry` package has been placed into the public domain. Thus, absolutely no licensing issues have to be considered. Figure 3.4 shows the classes defined for registry access.

Package com.ice.jni.registry

Class Summary	
HexNumberFormat	The HexNumberFormat class implements the code necessary to format and parse Hexidecimal integer numbers.
RegBinaryValue	The RegBinaryValue class represents a binary value in the registry (REG_BINARY).
RegDWordValue	The RegDWordValue class represents a double word, or integer, value in the registry (REG_DWORD).
Registry	The Registry class provides is used to load the native library DLL, as well as a placeholder for the top level keys, error codes, and utility methods.
RegistryKey	The RegistryKey class represents a key in the registry.
RegistryValue	The RegistryValue class represents a value in the registry.
RegMultiStringValue	The RegMultiStringValue class represents a multiple string, or string array, value in the registry (REG_MULTI_SZ).
RegStringValue	The RegStringValue class represents a string value in the registry (REG_SZ, and REG_EXPAND_SZ).

Figure 3.4: Classes of JNIRegistry [5]

The Java code for changing the computer name is very similar to the example of the jRegistryKey. Therefore it was not included here.

It is always a little dangerous to manipulate the registry directly. If a wrong key has been changed or deleted, the whole computer might get stuck. To avoid this, it is recommended to back up the registry before making any changes. So if the worst comes to the worst, the old state of the registry can be restored.
Another disadvantage is, that there is no standard which keys have to be modified. Also, there is no way to access the registry remotely. This would have to be implemented. Thus we will look at some other solutions before we decide.

3.2 XML

An easier solution would be to write an XML file with the concrete configurations and to store this to an appropriate directory, such as the text files in Linux. The only possibility seemed to be the .NET Framework as this supports a lot of XML and also has an XML parser.

The .NET Framework is an environment in Windows for the development, supply and design of XML Web services and other applications. It is focused on platform independence and network transparency. The .NET Framework is composed from a set of programming languages (C#, VB .NET...) that are completely object oriented. It has cross-language compatibility, meaning that .NET components can interact with each other regardless of the language in which they where written. For more detail about the .NET Framework see [7] and [6].

After a long research on the Microsoft homepage and [7] there was no solution found, how one can configure a Windows machine directly using XML. There exist configuration files which are text files using the XML syntax in the .NET Framework. On the first sight they seem perfect as there exist also machine configuration files. But they are not designated for configuring Windows resources. They are rather to configure the .NET Framework itself. This means configuring global settings for ASP.NET, remoting and global assembly cache. So the research has to be continued.

3.3 WMI

Having no solution to configure Windows machines using XML, I came upon the Windows Management Instrumentation (WMI). WMI is a component of the Windows operating system through which Windows resources can be accessed, configured and managed.
WMI is preinstalled in Windows 2003, Windows XP, Windows ME and Windows 2000. For Windows NT and Windows 98/95 WMI can be downloaded from [8] (WMI CORE 1.5 (Windows 95/98/NT 4.0)).
In Chapter 4, WMI will be treated in detail. We will now discuss the available Java API's for WMI.

3.3.1 Java API for WMI

WBEM Services

The WMI can also be accessed using Java. WBEM Services [9] seem to be a potential API. WBEM Services are an open source Java implementation of Web-Based Enterprise Management (WBEM) (see Chapter 4), developed by Sun Microsystems Inc. The source code is available under the Sun Industry Standards Source License (SISSL).
Several classes are provided to handle the Common Information Model (CIM) (see 4.2). After having tested WBEM Services, result was that one can not access any Win32 classes. The reason is that the implementation of WBEM Services does not include the necessary adapters to interact with WMI. WMI uses the Distributed Component Object Model (DCOM) to communicate with the Common Information Model Object Manager (CIMOM). In contrast the WBEM Services project uses HTTP. Thus WBEM Services can be used to connect to servers running WMI, but one can not get any information from WMI.

Jawin

Another Java API to access the WMI is Jawin [10]. The Java/Win32 integration project (Jawin) is a free, open source architecture for interoperation between Java and components exposed through Microsoft's Component Object Model (COM) or through Win32 Dynamic Link Libraries (DLLs). Jawin can be used to call any component that can be scripted in the Microsoft environment without writing any JNI code.
The fact that one can manipulate any information made available through WMI using scripts makes this approach interesting. An example of how to use Jawin is shown in Section 5.2, Figure 5.1.

3.4 Conclusion

We have two alternatives to configure a Windows machine remotely using Java. The first one is to write a server to manipulate the keys of the registry. The second one is to configure Windows with the Windows Management Instrumentation. The better solution is to configure the network settings using WMI.

In WMI we have just to know the classes that are responsible for the resource we want to manage. To access WMI, the structure of the example in Figure 5.1 can always be used. Only the WMI class we will access, the method we will execute and the input parameters need to be changed. This can be defined using XML.

We will access the WMI using Jawin. The disadvantage of Jawin is that it only runs on Windows machines. So the WMI can not be manipulated from a Linux machine. But it is still better and easier to configure Windows using Jawin than to do it through the registry, as we do not have to know where all the keys are stored. Furthermore it is very easy to configure Windows remotely. We can just write the hostname of the computer we want to access.

Chapter 4

Windows Management Instrumentation

The Windows Management Instrumentation (WMI) has been the central management technology since Windows 2000. It is a component of the Windows operating system through which Windows resources can be accessed, configured, managed and monitored using scripts. Through WMI, computers can be managed locally or remotely over a network. WMI uses the Distributed Component Object Model (DCOM) to handle remote calls.

WMI is the Microsoft implementation of the Web-Based Enterprise Management. WBEM is a standard of the Desktop Management Task Force (DMTF) to manage network and system resources across a network. WBEM is defined independently of protocol or management standards.

The core of WBEM is the Common Information Model (CIM), which shapes the managed resources of WBEM through object-oriented methods. CIM is a framework which describes both physical and logical objects. CIM is designed to complement existing management standards like the Simple Network Management Protocol (SNMP) or the Common Management Information Protocol (CMIP). WBEM provides an integration point through which data from all such sources can be accessed.

WBEM is an object-oriented approach. Every resource is represented by objects, which are combined in classes. The representation of a resource in WBEM is called a managed object.

The name Web-Based Enterprise Management is misleading, as it suggests that WBEM is about a web-based graphical user interface for the management of system information. However WBEM is simply an architecture with a programming interface, therefore neither tool nor application. Why it is called web-based keeps a mystery of the developers.

For more details about WMI and WBEM see [14].

4.1 WMI Architecture

The complicated part of working with WMI is not writing code, but to find out which classes are available to access the required resources. Understanding the WMI architecture is essential to know what classes WMI can manage as well as which methods and properties may be used with each class. According to [12], the WMI architecture consists of three main layers as shown in Figure 4.1. These will be explained in the subsections.

- Managed Resources
- WMI Infrastructure
- Consumers

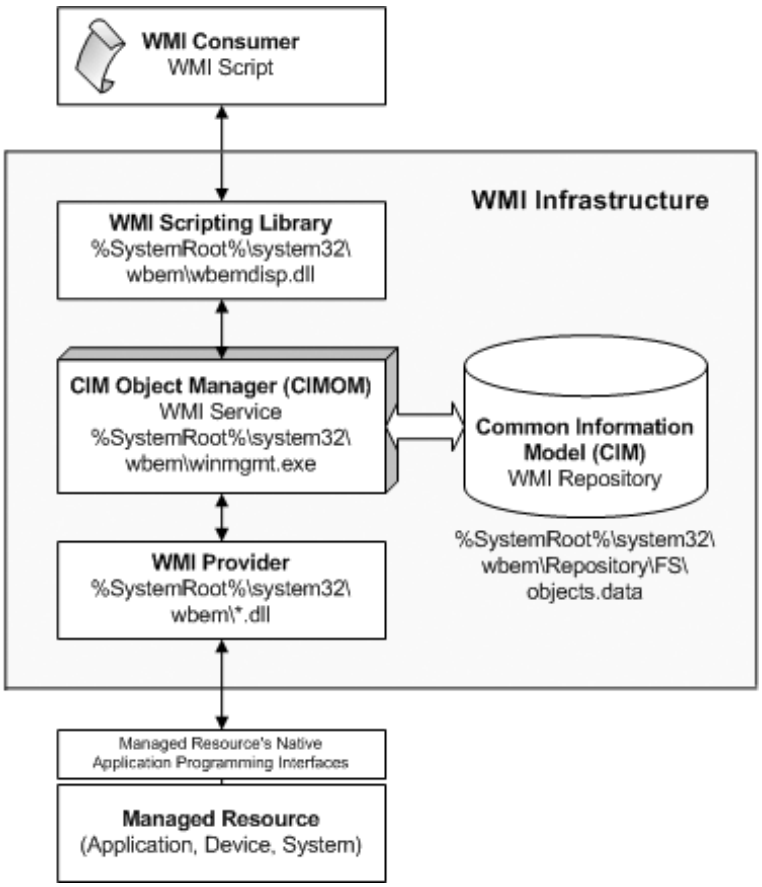

Figure 4.1: WMI Architecture [12]

4.1.1 Managed Resources

We will begin at the lowest layer as this is where resources are located. A managed resource is an arbitrary logical or physical component which is made available and manageable through WMI. Windows resources that can be managed using WMI include the computer system, disks, event logs, file systems, network components, printers, processes, security (authentication/ authorization) and many more.

4.1.2 WMI Infrastructure

The middle layer is the WMI infrastructure. WMI consists of three main components, which are the Common Information Model Object Manager (CIMOM), the CIM repository and WMI providers. Together these three WMI components build the infrastructure through which configuration and managed data is defined, exposed, retrieved and accessed. The fourth component, small but necessary, is the WMI scripting library.

WMI Providers

The providers are one of the main components of the WMI architecture. They facilitate the communication between WMI and a managed resource. Providers request information from and send instructions to WMI managed resources. WMI providers communicate with their appropriate managed resources using the managed resources native API's, and communicate with the CIMOM using WMI programming interfaces.
Software developers have the possibility to develop and integrate add-on providers, to make the management functions available uniquely to their products.

CIMOM

The Common Information Model Object Manager allows the interaction between consumer and provider. All requests from the WMI pass through the CIMOM. In Windows XP and Windows Server 2003 the WMI service, winmgmt.exe, adopts the role of the CIMOM.
Beside the common interface through which consumers access WMI, the CIMOM provides core services as the provider registration, request routing, remote access, security (authorization), query processing and event processing.
Applications call into the CIMOM to perform management-related tasks. The CIMOM calls the essential provider and class information from the CIM to handle the consumer requests. The CIMOM uses the information, obtained from the CIM, to pass the requests to the appropriate provider.

CIM Repository

The idea of WMI is that configuration and management information from different sources can be represented uniformly using a schema. The Common Information Model is the schema which models the managed environment and defines all data elements provided by WMI. The schema is based on the DMTF Common Information Model Standard (for more details see [13]).
The CIM consists of classes. A class is a model of a resource managed by the WMI. CIM classes usually represent dynamic resources. This means instances of resources are not stored in the CIM, but they are dynamically called by a provider. The reason for this is that the operational state for most managed resources changes frequently and therefore must be read on-demand to ensure to get the actual information.

CIM classes are hierarchically structured, whereas child classes inherit from parent classes. Some central and general base classes are maintained by the DMTF, from which software developers, such as Microsoft developers, derive and create system- or application-specific extension classes. It is important to understand the basic structure of CIM, as well as to navigate and interpret its content for writing WMI-based scripts. In section 4.2 we will discuss the CIM and its classes in detail.

WMI Scripting Library

The WMI scripting library contains some automation objects, through which script languages can access the WMI infrastructure. The automation objects provide a continuous and uniform scripting model for the WMI. Once it has been understood how to call one managed resource type using the scripting library, with the same steps other WMI managed resources can be called easily.
We will treat the scripting API for WMI in section 4.3.

4.1.3 Consumers

Consumers form the top layer. A consumer can be a script, an enterprise management application, a web-based application or another administrative program, which accesses and controls management information that are available through the WMI infrastructure.

More information about the WMI architecture can be found in [12], Part 1.

4.2 Common Information Model

As mentioned in the previous section, the Common Information Model (CIM) repository is the schema which represents configuration and management information from different sources uniformly. The CIM is a model of the hardware, operating system and software a computer consists of. It is the data model for WMI.

Although the CIM repository is able to store some data, its primary purpose is to model the managed environment. The CIM does not store the voluminous management information it defines. Rather, most data is dynamically retrieved on-demand by a WMI provider.

Figure 4.2 shows the internal structure of the CIM repository schematically. As one can see, the CIM uses classes to create the data model. The CIM contains about 5000 classes, not only the eleven illustrated in the diagram. The number of classes is negligible in the context of WMI scripting as particularly the concept must be understood. But it is important to know that the CIM repository is the class store which defines the WMI manageable environment and every managed resource provided through WMI.

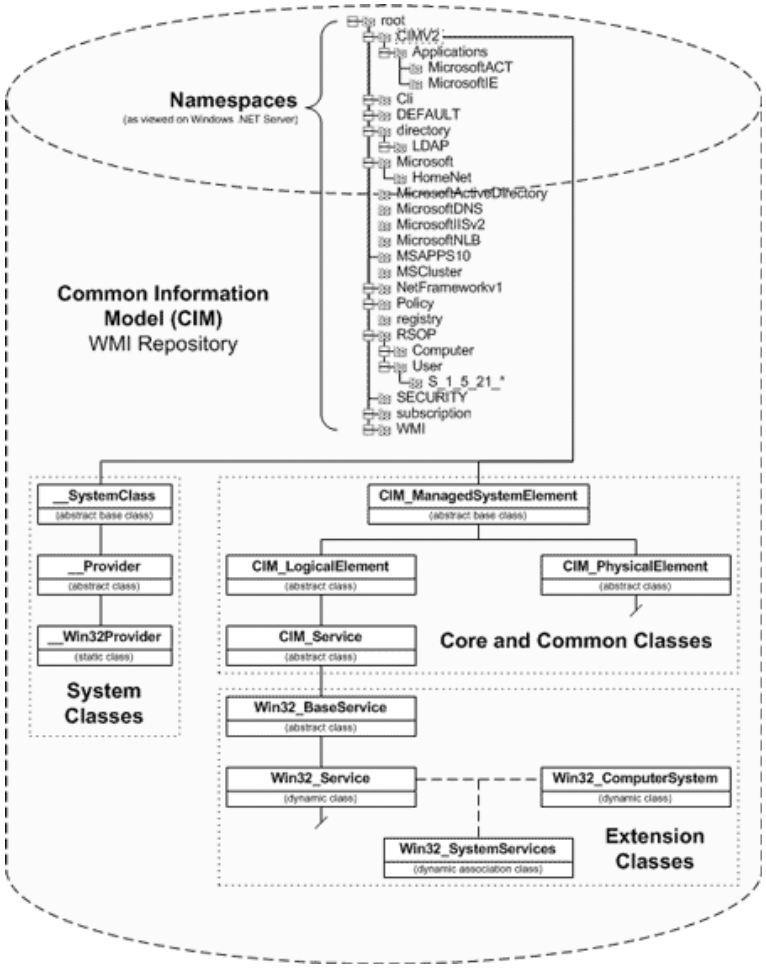

Figure 4.2: Structure of the CIM Repository [12]

Figure 4.2 illustrates three important CIM concepts to navigate and interpret the WMI schema successfully.

- The CIM repository is divided into multiple namespaces
- Each namespace can contain one or more of the following class categories: system classes, core and common classes and/or extension classes
- There are three primary class types: abstract, static and dynamic

These CIM concepts are examined in detail in the following.

4.2.1 Namespaces

CIM classes are divided into namespaces. Namespaces control the scope and visibility of managed resources class definitions. Each namespace contains a group of related classes, which represent a certain technology or management area. Within a namespace, all classes must have a unique class name. Classes within a namespace can not be derived from classes in another namespace. Therefore identical system classes as well as core and common classes are defined in more than one namespace.
Most classes modelling Windows managed resources are in the `root/cimv2` namespace. However, this is not the only namespace to be considered when writing scripts.
How do namespaces influence WMI scripts? In a first step each script establishes a connection to the namespace (see Figure 5.1). Adding the target namespace to the connection, the CIMOM obtains information where the class definition of the managed resource can be found in the CIM. If the target namespace is not specified, the script connects to the default namespace (attention: this is not the same as root/default!). When the class definition of the managed resource can not be found in the default namespace, an error occurs. To avoid such errors due to invalid namespaces, it is better to always specify a target namespace.

4.2.2 Class Categories

As shown in Figure 4.2 there are three general categories of classes used to construct the CIM: system classes, core and common classes and extension classes.

System Classes

System classes are classes which support the internal WMI configuration and operation. By browsing the CIM system classes can be recognized easily by the two underscores preceding the name of the system class. In Figure 4.2 `__SystemClass`, `__Provider` and `__Win32Provider` are system classes.
System classes are either abstract or static. Abstract system classes are blueprints to derive from other abstract or static system classes. Static system classes define WMI configuration and operational data which are physically stored in the CIM repository.

Core and Common Classes

The core and common classes have two functions: Primary, they represent the abstract classes from which software developers derive and create extension classes. Secondary, they define resources which share certain management areas, but which are independent of a certain technology or implementation. The DMTF defines and manages a set of core and common classes, which can be recognized by the prefix `CIM_`. The four classes `CIM_ManagedSystemElement`, `CIM_LogicalElement`, `CIM_Service` and `CIM_PhysicalElement` shown in Figure 4.2 are core and common classes.

Extension Classes

Extension classes are technology-specific classes created by system and application software developers. The four classes `Win32_BaseService`, `Win32_Service`, `Win32_SystemServices` and `Win32_ComputerSystem` illustrated in Figure 4.2 are Microsoft extension classes. But it should not be concluded that all Microsoft extension class names start with `Win32_`. There are also some that do not. Extension classes are the primary category of classes that will be used for writing WMI scripts.

4.2.3 Class Types

Classes are the basis of the CIM repository. CIM classes are hierarchically structured whereas child classes inherit methods, providers and qualifiers form parent classes. For example the dynamic class `Win32_Service` is inherited from the abstract class `Win32_BaseService`, which is inherited from the abstract class `CIM_Service` and so on (see Figure 4.2). It is the sum of classes in the class hierarchy that finally defines a manageable resource. There are three primary class types: abstract, static and dynamic.

Abstract Classes

An abstract class is a blueprint used to define a new class. Abstract classes act as base classes for other abstract, static or dynamic classes. Nearly every WMI managed resource class definition is based on one ore more abstract classes.

Static Classes

A static class defines the data physically stored in the CIM repository. Static classes consist of instances, which are stored in the CIM repository. The instances of static classes are retrieved directly from the CIM. They do not use a provider.

Dynamic Classes

A dynamic class models a WMI managed resource which is dynamically called by a provider. The dynamic class type is mostly used for the definition of extension classes. Dynamic classes are the class type usually used in WMI scripts.

Association Classes

A fourth class type is the association classes. An association class is an abstract, static or dynamic class that describes the relationship between two classes or managed resources. The `Win32_SystemServices` class shown in Figure 4.2 is an example of a dynamic association class. It describes the relationship between a computer and the services running on the computer.

4.2.4 Class Structure

As already mentioned, all hardware and software resources that are manageable through WMI are defined by a class. All class definitions of a manageable resource follow a well defined structure and syntax as shown in Figure 4.3. Every class definition is made up of properties, methods and qualifiers.

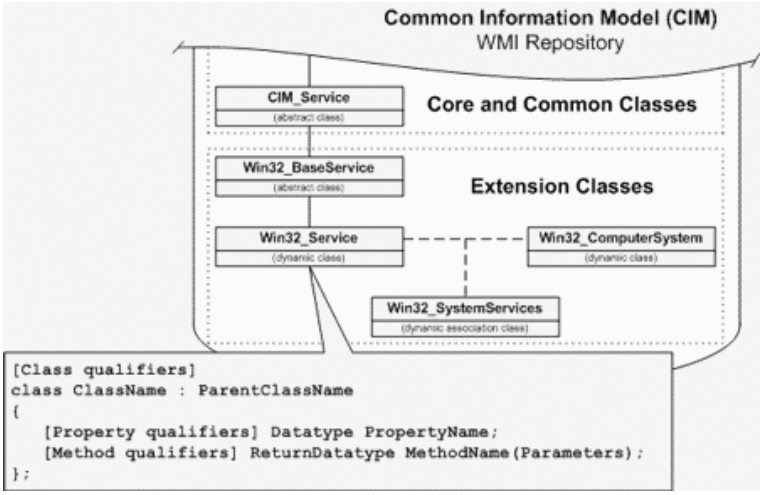

Figure 4.3: Structure of a Managed Resource Class Definition [12]

Properties

A property describes a manageable resource. Classes use properties to describe several aspects as identity, configuration and state of a managed resource. For example `Win32_Service` has a name, display name, description, start-up type and status. Every property has a name, type and optional property qualifiers.

Methods

Methods perform actions on manageable resources. Services can be started, stopped, paused and continued. Each method has a name, return type, optional parameters and optional method qualifiers.

Qualifiers

Qualifiers provide additional information about classes, properties and methods they apply to. For example the type of `Win32_Service` is defined by the `Dynamic` qualifier of the class. When writing scripts which retrieve not only information, but modify properties or call methods, qualifiers become increasingly important as they define the operational characteristics of the property that will be modified.

- **Class Qualifiers**: provide operational characteristics about a class (`Abstract`, `Dynamic`, `Association` qualifier; `Provider` qualifier; `Privileges` qualifier)
- **Property Qualifiers**: provide information about each property (`CIMType` qualifier; `Read/Write` qualifier; `Key` qualifier)
- **Method Qualifier**: provide information about each method (`Implemented` qualifier; `ValueMap` qualifier; `Privileges` qualifier)

There are lot more qualifiers than mentioned above. For the whole list see WMI qualifiers [15]. The information of this section is taken from [12], Part 2. More details about the Common Information Model can be found there.

4.3 Scripting API for WMI

The WMI scripting library provides a uniform set of controls, in the form of automation objects, that allow managing and accessing WMI managed resources. It does not make any difference whether writing code to manage computers, event logs, operating system, processes or services; the objects of the WMI scripting library always work the same.
The consistency of the automation objects is best communicated through a finite set of tasks one can perform using the WMI scripting library. Altogether seven basic script types can be created using the WMI scripting library:

1. Retrieve the instances of a WMI managed resource
2. Read the properties of a WMI managed resource
3. Change the properties of a WMI managed resource
4. Call a method of a WMI managed resource
5. Create a new instance of a WMI managed resource
6. Delete an instance of a WMI managed resource
7. Subscribe to events for monitoring the creation, modification and/or deletion of a WMI managed resource.

The seven basic script types are like script templates. They can be used to manage any WMI managed resource. Once a template to manage one type of WMI managed resource has been understood, this can be adapted easily to hundreds of other WMI managed resources.

4.3.1 WMI Scripting Library Object Model

Now that we know that the WMI scripting library is the system control of the whole WMI infrastructure, we will look at this more precisely. The WMI scripting library is implemented in a single automation component named wbemdisp.dll, which is stored in the directory C:/WINDOWS/ System32/WBEM in Windows XP. Totaling, the WMI scripting library consists of twenty-four automation objects. A part of them is illustrated in Figure 4.4. But not all twenty-four automation objects have to be known in detail. Many scripts can be created with the basic understanding of just four or five of the objects shown Figure 4.4.
It is important to understand the relationship between the automation objects of the WMI scripting library and the class definitions of managed resources that are stored in the CIM repository. As illustrated in Section 4.2, class definitions for managed resources are blueprints for the computer resources that are exposed through WMI. Besides the resources that can be managed, the blueprints also define the methods and properties unique for each managed resource.

On the other hand, the WMI scripting library contains a set of general scripts for automation objects. These scripts are used to authenticate and connect to WMI and subsequently access instances of WMI managed resources. When an instance of a WMI managed resource is retrieved, one can access the methods and properties defined by the class definition of managed resources.

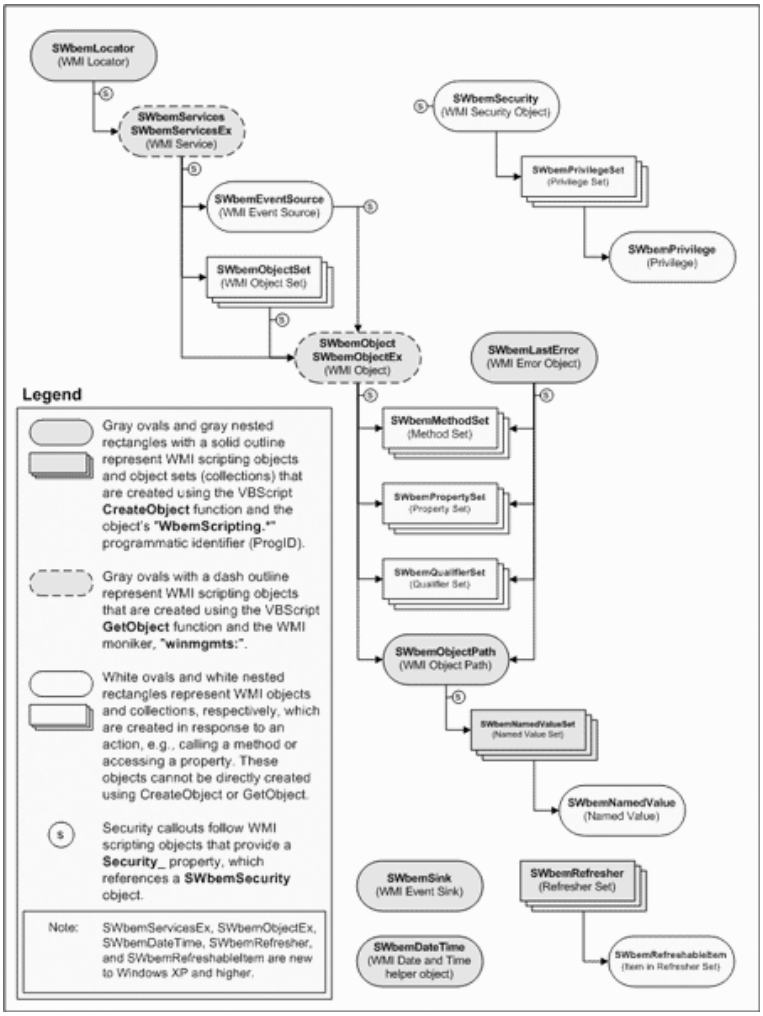

Figure 4.4: WMI Scripting Library Object Model, wbemdisp.dll [12]

The lines in Figure 4.4 point out to the object which is obtained by calling a method of the originating object. For example, calling the `ConnectServer` method of `SWbemLocator` returns a `SWbemServices` object. Calling the `ExecNotificationQuery` method of `SWbemServices` returns a `SWbemEventSource` object. On the other hand, calling the `SWbemServices ExecQuery` method returns a `SWbemObjectSet` collection. Finally, calling the `Get` method of `SWbemObjectSet` returns a `SWbemObject` object.

- **SWbemServices** is the object that represents an authenticated connection to a WMI namespace on a local or remote computer.
- **SWbemObjectSet** is a listing of null or more `SWbemObject` objects. Each `SWbemObject` in a `SWbemObjectSet` can represent either an instance of a WMI managed resource or an instance of a class definition.
- **SWbemObject** is the object with several identities, which imitates the resource one is managing. For example, if instances of a `Win32_Process` managed resource are retrieved, `SWbemObject` adopts an identity that is modeled after the class definition of `Win32_Process`. Otherwise, if instances of a `Win32_Service` managed resource are retrieved, `SWbemObject` adopts an identity modeled about the `Win32_Service` class.
  The `SWbemObject` object is able to transform itself to an arbitrary manageable resource that is defined in the CIM. Once we know how to connect and retrieve instances, everything is a `SWbemObject`.

These automation objects are part of nearly all WMI scripts. For more information about the WMI scripting library see [12], Part 3. Details about specific objects, methods or properties of the scripting library will be found on [16].

4.4 WMI Query Language

WMI supports a large and powerful query facility for retrieving instances of managed resources. For example, a WMI query can request only services with the `StartMode` auto which are in the state `stopped`. So, WMI queries provide an efficient mechanism for retrieving instances, since they return only those instances and properties which correspond to a certain criteria. Additionally, the queries are treated on the target computer and not on the source computer where the script is running. Therefore, WMI queries can essentially reduce the amount of network traffic.
WMI queries are created using the WMI Query Language (WQL). WQL is a subset of the Structured Query Language (SQL) with minor semantic changes to support WMI. The query string defines criteria that must be fulfilled to result in a successful match. After the definition of the query string the query is passed to the WMI service using the `ExecQuery` method of `SWbemServices`. Instances of the managed resource that correspond to the query are returned to the script as a `SWbemObjectSet` collection.
Details about querying with WQL can be found on [12], Part 3 .

4.5 Network Configuration using WMI

The Windows Management Instrumentation enables system administrators to automate many common tasks involved in TCP/IP network clients. Using WMI, the following network tasks can be managed:

- Retrieve and manage TCP/IP client settings
- Work with remote hosts
- Configure DHCP and static IP addresses
- Manage DNS client settings
- Work with NetBIOS and WINS

For the work with the network and network configuration the following WMI classes are important:

- **Win32_PingStatus**: represents the values returned by the standard ping command.
- **Win32_NetworkAdapterConfiguration**: represents the attributes and behaviours of a network adapter.
- **Win32_NetworkAdapter**: represents a network adapter on a Windows system.

4.5.1 Ethernet

To configure the Ethernet card with the settings from Verinec, the WMI class `Win32_Network AdapterConfiguration` of the namespace `root/cimv2` is used. This class represents the attributes and behaviours of a network adapter. It includes extra properties and attributes that support the management of the TCP/IP and Internetworking Packet Exchange (IPX) protocols, which are independent of the network adapter.
The following methods are used to configure the Ethernet card:

- **EnableDHCP**: This method enables the Dynamic Host Configuration Protocol (DHCP) for service with this network adapter. DHCP allows IP addresses to be dynamically allocated. This method has no parameters.
- **EnableStatic**: This method enables static TCP/IP addressing for the target network adapter. This method has two parameters: `IPAddress` which lists all the static IP addresses for the current network adapter and `SubnetMask` that complement the values in the `IPAddress` parameter. Both parameters are required.
- **SetDNSServerSearchOrder**: This method uses an array of string elements to set the ordered list of DNS servers. The only parameter of this method is the required `DNSServerSearchOrder` which is a list of server IP addresses to query for DNS servers.

- **SetGateways**: This method specifies a list of gateways for routing packets to a subnet which is different from the subnet that the network adapter is connected to. This method only works when the Network Interface Card (NIC) is in static IP mode. This method has two parameters: `DefaultIPGateway` (required) is a list of IP addresses to gateways where network packets are routed and `GatewayCostMetric` (optional) which assigns a value that ranges from 1 to 9999, which is used to calculate the fastest and most reliable routes. The default value for a gateway is one.

The return values of these methods are zero for a successful completion when no reboot is required, one for a successful completion if a reboot is required. Any other number produces an error.
The `Win32_NetworkAdapterConfiguration` class with all its properties and methods is explained in detail on [17].

4.5.2 Dial-Up Modem

Another hardware component to be configured in Verinec is the dial-up modem. There are two `Win32` classes in the namespace `root/cimv2` provided to handle telephone modems:

- **Win32_POTSModem**: represents the services and characteristics of a Plain Old Telephone Service (POTS) modem on a Windows system.
- **Win32_POTSModemToSerialPort**: relates a modem and the serial port the modem uses.

The problem of these classes is that they have no or no suitable methods, and the properties are all read-only. It was tried to write the properties anyway, but without success. As a result there is no way to modify the properties and so they can not be used to configure a dial-up modem.

There is another interesting class defined in the `root/RSOP/Computer` namespace:

- **RSOP_IEConnectionDialUpCredential**: represents the settings used by the `RasDial` function when establishing a dial-up (remote access) connection to the Internet using Microsoft Internet Explorer.

Actually this class establishes a connection for the Microsoft Internet Explorer. But probably it would also work for other browsers. But we have the same problem as above. The class does not support any methods and the properties are all read-only.

Although questions were posted to different forums, so far no solution was found, how one can configure a dial-up modem using WMI. This still needs investigation.

4.6 User Account Management

A planned service to handle in Verinec is the user account management. This section presents how the user account management is handled on Windows machines.
There are two different possibilities to manage user accounts in Windows XP Professional, depending on whether the computer is member of a workgroup or of a network domain. As in bigger networks computers are part of a network we will look at the approach where computers are members of a network domain.
On a computer which is part of a network domain, a user must be a member of at least one group. By adding a user to a group, the user gets all the permissions and rights assigned to that group. A user can be assigned to the following groups:

- **Administrators**: Members of this group have the largest amount of default permissions and the ability to change their own permissions. They can configure and mange the system.
- **Backup Operators**: Members of this group can perform backup and restore operations on the computer, regardless of any permissions. But they can not change security settings.
- **Guests**: Allows users to log on the computer without the need for a separate account for each user. There is no password for the guest account.
- **Help Service Group**: Members of this group can use applications to help diagnose system problems.
- **Network Configuration Operators**: Members of this group are able to provide limited administrative functions, such as assigning IP addresses.
- **Power Users**: Members of this group lie somewhere between administrators and users. They can create new user accounts, but can only modify the accounts they have created. They can install and modify applications and they can install local printers.
- **Remote Desktop Users**: Members of this group are allowed to log on remotely.
- **Replicator**: Members of this group are allowed to replicate files across a domain.
- **Users**: Members of this group can perform most common tasks, such as running applications. They have limited access to the file system and have read and write permissions only on their own profile.

So, finally every user account belongs to a group, which belongs on its part to a domain. Of course, a user account needs a user name as well as a password for authentication.
For more information about the user account management see [18].

4.6.1 User Account Management in WMI

WMI provides a couple of classes for the representation of user accounts:

- **Win32_Account**: contains information about user accounts and group accounts known to the Windows system.
- **Win32_Group**: represents data about a group account. A group account allows access privileges to be changed for a list of users.
- **Win32_GroupInDomain**: identifies the group accounts associated with a Windows domain.
- **Win32_GroupUser**: relates a group and an account that is a member of that group.
- **Win32_UserAccount**: contains information about a user account on a Windows operating system.
- **Win32_SystemAccount**: represents a system account. The system account is used by the operating system and services that run under Windows.
- **Win32_UserInDomain**: relates a user account and a Windows domain.

The WMI classes `Win32_Group` and `Win32_UserAccount` only provide one method `Rename` that allows the renaming of the group respectively the user account. All other classes do not provide any methods and the properties are read-only. Just the `Win32_UserAccount` class has some writeable properties. Though these are not enough to handle the whole user account management.
The Microsoft TechNet Script Center [19] provides a lot of other scripts for the management of users (follow the path TechNet Home > Script Repository > Active Directory > UserAccounts). Probably there is a possibility to fit them in Jawin as with Jawin one can do all the things one can do using scripts. However, this still needs investigation.

Chapter 5

Jawin

As already mentioned, the Java / Win32 integration project (Jawin) is an open source architecture for interoperation between Java and components exposed through Microsoft's Component Object Model or through Win32 Dynamic Link Libraries.
Jawin can be used for interaction with scriptable applications. One can use Jawin to call scriptable logic components such as Microsoft's COM-based XML parsers and tools. Jawin can also be used to access Win32 API features such as the Windows registry, security API's and the event log. Jawin allows the Java applications to call any COM- and DLL-based code, without writing any JNI code. Using Jawin, one can call any component that can be scripted in the Windows environment.

5.1 Getting Started with Jawin

Jawin requires Java Development Kit (JDK) 1.3 or higher and can only be used on Windows machines with COM-support. All versions since Windows 95 have some kind of COM-support. Before writing code using Jawin, the single Jawin library must be loaded. No other native code is necessary. Jawin itself is a standard JNI library. The Jawin native code is stored in the jawin.dll file, which can be found on the CD under WindowsTranslator/lib. To make the Jawin library visible to the application, the java.library.path property must be set to point to the location of jawin.dll. Then the jar file jawin.jar that is stored on the CD under WindowsTranslator/java/jars must be put on the classpath of the project.
For an easier use of the WMI API, we have to generate Jawin stubs for the native objects. The Jawin Type Browser makes it possible to automatically generate the necessary Java stubs. No COM knowledge is required to use it successfully. How stubs can be created using the Jawin Type Browser is described in detail in [10]. In order to use the WMI API, stubs from the WMI scripting type library wbemdisp.tlb have to be created. This library will be exported form the C:/WINDOWS/SYSTEM32/WBEM directory. The generated stubs are stored as wmi.jar and can be found on the CD under WindowsTranslator/java/jars.

5.2 WMI Scripts in Jawin

Every WMI script has three steps in common:

1. Connecting to WMI services
2. Retrieve instances of WMI managed resources
3. Administrate WMI managed resources

The example in Figure 5.1 shows how a static IP Address can be assigned to a network adapter following these steps.

WMI Script in Jawin

```
import wmi.script.*;
import org.jawin.*;

public class StaticIP {
  public static void main(String[] args) throws COMException {

    String host = ".";
    String username = "";
    String password = "";
    String namespace = "root/cimv2";
    String query ="SELECT * FROM Win32_NetworkAdapterConfiguration WHERE Index = 1";
    String[] IPAddresses = {"134.21.54.85"};
    String[] SubnetMasks = {"255.255.255.0"};
    String MethodName = "EnableStatic";

    try{
      ISWbemLocator locator = new ISWbemLocator ("WbemScripting.SWbemLocator");
      ISWbemServices wbemServices = locator.ConnectServer
          (host, namespace, user, password, "","",0,null);
      ISWbemObjectSet wbemObjectSet = wbemServices.ExecQuery(query, "WQL", 0, null);
      int sizeOfObjectSet = wbemObjectSet.getCount();
      DispatchPtr[] results = new DispatchPtr[sizeOfObjectSet];
      IUnknown uk = wbemObjectSet.get_NewEnum();
      IEnumVariant ev = (IEnumVariant) uk.queryInterface(IEnumVariant.class);
      ev.Next(sizeOfObjectSet, results);
      for (int i=0; i<results.length; i++){
        ISWbemObject wbemObject = (ISWbemObject)
            results[i].queryInterface(ISWbemObject.class);
        ISWbemMethodSet methodSet = (ISWbemMethodSet) wbemObject.getMethods_();
        ISWbemMethod method = methodSet.Item(MethodName, 0);
        ISWbemObject inParamClass = method.getInParameters();
        ISWbemObject inParamInstance = inParamClass.SpawnInstance_(0);
        inParamInstance.put("IPAddress", IPAddresses);
        inParamInstance.put("SubnetMask", SubnetMasks);
        ISWbemObject outputParam = null;
        outputParam = wbemObject.ExecMethod_
            (MethodName, inParamInstance, 0, null);
      }
    }
    catch (COMException e){
      System.out.println("COMError: " + e.hresult);
    }
  }
}
```

Figure 5.1: Assign a Static IP Address to a Network Adapter using Jawin

In a first step, some variables are declared which make the code more readable. The dot "." of the variable `ComputerName` indicates the local computer. If a remote computer will be accessed, the dot can be changed to the host name of the computer that will be accessed. As we will manage the local host, the variables `username` and `password` are empty strings. User credentials can only be used for remote connections. On the local machine, WMI can only be used with the rights of the user running the application.
The first thing the WMI script does is establishing a connection to the WMI. For this purpose, we have to create an `ISWbemLocator`. By calling the `ConnectServer` method we connect to the `namespace` on the computer specified in the `host` parameter. The target computer must have WMI installed.
In a second step, the script retrieves the instances of the `Win32_NetworkAdapterConfiguration` class where the `Index` equals `"1"` using the `ExecQuery` method. The index number specifies which network adapter is addressed. At this point, this is the Ethernet card of the first slot.
The third step is to manipulate the configuration of the network adapter. As the `ExecQuery` method returns a `WbemObjectSet` collection, we have to visit each element of this collection. In this example the size of the `WbemObjectSet` is only one, but this does not affect the script. With the `getMethods_` method we get a collection of all the methods of the `Win32_NetworkAdapter Configuration` class. Then with the `Item` method we get the method we will execute. In this case this is `EnableStatic`. This method has two parameters, so we have to fill in the input parameters. For this, we first get the class of the input parameters using the `getInParameters` method. Then with the `SpawnInstance_` method an instance of this class will be created. Now we fill in the parameters with their values using the `put` method. The input parameter `IPAddress` will be filled with the value of `IPAddresses` and the input parameter `SubnetMask` will be filled with the value of `SubnetMasks`. Finally, the method will be executed using `ExecMethod_` with the method name and the input parameters.
The types starting with `ISWbem` describe WMI scripting objects. They were treated in detail in Section 4.3.

Now we have the basic structure of a script. If we will access other WMI managed resources, we can just change the query, the method name and the input parameters. The rest of the script keeps the same.

Chapter 6

Windows Translator in Detail

The base of Verinec is the abstract network definition described using the XML syntax. A network consists of nodes. Nodes may be workstations, servers or other devices like switches, routers or hardware firewalls. Every node contains a hardware section describing the physical part and a service section, where all the services supported by this node are defined. Note, that in the translation context, the children of the hardware section (Ethernet, serial, WLAN) are treated like services. Figure 6.1 shows a simple node of a network. More information about the network definition can be found on [1] and [2].

A simple node

```
<node hostname="pc01">
  <hardware>
    <etherent name="First Ethernet Card">
      <ethernet-binding name"eth0" id="xyz">
        <nw id="i01" address="134.21.54.85" subnet="255.255.255.0"
         gateway="134.21.54.1" type="ip">
          <nw-dnsserver ip="134.21.14.50" />
          <nw-dnsserver ip="134.21.1.31" />
        </nw>
      </ethernet-binding>
    </ethernet>
  </hardware>
  <services>
    <!-- Service Configurations -->
  </services>
</node>
```

Figure 6.1: An Example of a Simple Node

The objective of the translator is to translate the abstract network configuration into system specific configuration. Every machine provides several services and for each service a particular translator has to be written. As a system can have more than one implementation for a service and different operating systems handle services differently, there might be many implementations for the same service. Metadata contains the information which translator has to be used.

6.1 Translation Process

In a first step, the translator gets the appropriate translation XSLT and restriction XSLT from the repository. A type tag specifies which translator and restrictor to use with a service. If there are some unsupported features for the translator selected, the restrictor will create warnings to the user. Then the translation begins. The proper translator is chosen from the repository and applied to the node. Finally the produced configuration data is distributed to the specified system. Figure 6.2 illustrates this process.

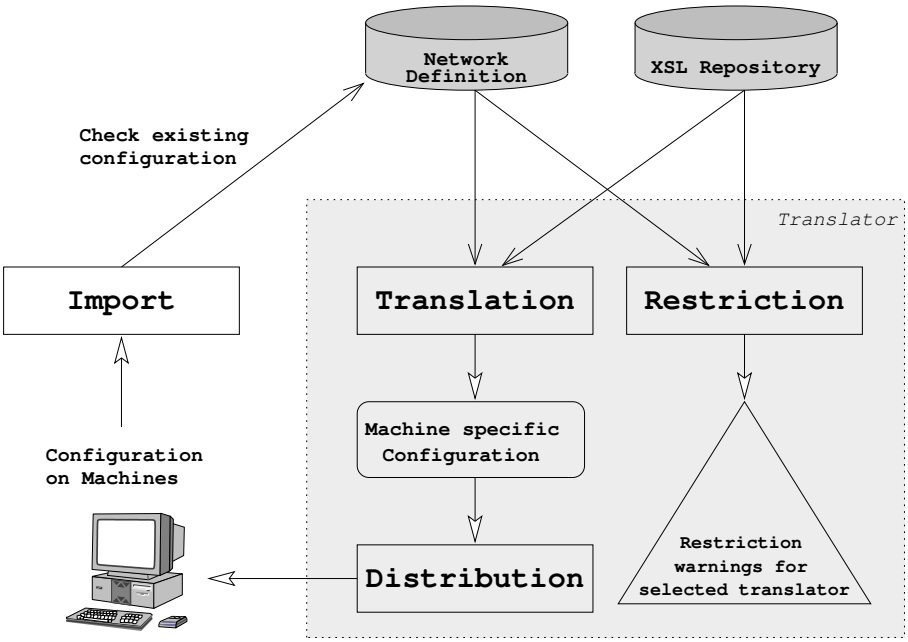

Figure 6.2: The Translation Process in detail [1]

6.1.1 XSL Repository

Every translation is composed of two XSLT documents which are stored in the repository: translator and restrictor. The translator XSLT translates the abstract configuration into machine specific configuration. The restrictor XSLT produces warnings to the user if some data can not be translated using the appropriate translator. Even if the translator is able to translate the whole configuration, a restrictor file is needed. In this case an empty result can be produced.
Both, the translation XSLT and the restriction XSLT, are identified by the service name and their own name. Both files must have the same name, usually the name of the service implementation with the extension .xsl. For example a service name can be `ethernet` and an implementation name can be `windows-xp` (see Figure 6.3).
The type tag specifies, which translator has to be used with a service. The prefix `tr` is usually used for the translation namespace. Figure 6.3 shows a type definition for the service `ethernet` running in a Windows XP machine. The translator used for this service is named `windows-xp`. There might be additional service tags, for other services, within a type.

Node type definition

```
<tr:type name="windows-xp" id="typ01">
  <tr:service name="ethernet" translation="windows-xp" />
</tr:type>
```

Figure 6.3: An Example of a Node Type Definition

6.1.2 Restriction

Sometimes an implementation of a service might not support all the features that can be configured in Verinec. For this case a restictor XSLT can output warnings for unsupported features. Restrictions are also produced if the schema is not valid or if something is missing to configure the feature correctly.
For example, Windows does not support the configuration of peerdns or bootp of an Ethernet card, which are configurable in Verinec's abstract configuration XML. So the restrictor `windows-xp.xsl` produces warnings if someone tries to set these features.

6.1.3 Translation

The translator is responsible to transform the abstract configuration into machine specific configuration using XSLT. The resulting documents contain the data to be distributed to the target system.
For Windows machines, the translator creates a `result-wmi` file. This is an XML file that contains the data needed to configure the services through the Windows Management Instrumentation. Figure 6.4 shows an example of a configuration output for an Ethernet card.

Configuration Output

```
<configuration xmlns="http://diuf.unifr/.../verinec/configuration"
            xmlns:vn="http://diuf.unifr.ch/.../verinec/node"
            xmlns:tr="http://diuf.unifr.ch/.../verinec/translation">
  <service name="ethernet">
    <tr:target xmlns="http://diuf.unifr.ch/.../verinec/node" name="target01">
      <tr:wmi host="pc01" />
    </tr:target>
    <result-wmi>
      <!--***Ethernet card configured by VeriNeC: ethernet 0***-->
      <wmi class="Win32_NetworkAdapterConfiguration" namespace="root/cimv2"
                wqlKey="Index" wqlKeyValue="1">
        <method name="EnableStatic">
          <param name="IPAddress" type="ArrayOfString">
            <param-value value="134.21.54.85" />
          </param>
          <param name="SubnetMask" type="ArrayOfString">
            <param-value value="255.255.255.0" />
          </param>
        </method>
      </wmi>
    </result-wmi>
  </service>
</configuration>
```

Figure 6.4: Configuration Output for an Ethernet Card in Windows XP

The result-wmi has to satisfy the configuration schema. This schema is described in detail in the next subsection.

Result-WMI Schema

Figure 6.4 shows how the `result-wmi` is structured. The `result-wmi` schema is part of the configuration schema. It defines which data the XML file must contain for a successful distribution. This includes the WMI methods with the corresponding parameters as well as some additional information like the WMI class used and the namespace where this is stored.
The `class` attribute specifies the name of the WMI class one want to access. The `namespace` attribute corresponds to the WMI namespace where the class is stored. The `wqlKey` and `wqlKeyValue` atrributes define the name of the property that forms the basis of the WQL WHERE clause and the appropriate value.
The `method` element specifies the method to be executed with the corresponding parameters. So far only parameters of the type `ArrayOfString` are supported.
Each `result-wmi` has one or more `wmi` elements. Every `wmi` element can have one or more `method` elements. A `method` element has zero or more `param` elements. Each `param` element can have one or more `param-value` elements. If there is more than one `param-value`, they are passed as an array to WMI.

6.1.4 Distribution

To distribute the configuration data on the target machine, Verinec has to know which distributor to use. This information is stored in the `tr:target` tag. Targets can either be defined globally in the nodetype or locally for a node. Figures 6.5 and 6.6 shows two possibilities of targets for the configuration of a Windows machine.

Target without User Credentials

```
<tr:target name="target01">
  <tr:wmi host="pc01" />
</tr:target>
```

Figure 6.5: Target for Local or Remote Connections

Target with User Credentials

```
<tr:target name="target02">
  <tr:wmi host="pc02" username="me@unifr" password="hello" />
</tr:target>
```

Figure 6.6: Target for Remote Connections

The target defined in Figure 6.5 can be used for local or remote connections. User credentials can not be used for local connections. For the connection to a remote machine they are optional. If the user credentials are not specified, the data of the user currently logged on is used. The target defined in Figure 6.6 can only be used for connections to remote machines. If the `username` attribute is defined, also the `password` attribute must be specified. But the `password` attribute can be defined without defining the `username` attribute. In this case the user currently logged on is used. The username can be in the form of either `user` or `user@domain`. The `host` attribute is always required. It defines which machine will be configured. The specification of the target is defined in the translation schema.
The `tr:target` tag tells the translator module which distributor has to be used. In this case this will be the suitable distributor for `tr:wmi` which is `DistWMI.java`. This distributor was implemented to configure Windows machines using the Windows Management Instrumentation. The aim of `DistWMI.java` is to distribute the data of the `result-wmi` document to a specified system. The basic structure of `DistWMI.java` is the same as shown in Figure 5.1. The namespace, method name, parameter name and value, as well as the data of the query are stored in the `result-wmi`. The host and user credentials are defined in the `tr:target` tag. As `result-wmi` is an XML file, its data can be retrieved using the Java Document Object Model (JDOM). The distributor outputs an exception, if an error occurs while executing the method. Otherwise the configuration was successful.
So far, only the parameter type `ArrayOfString` is supported. There exist other parameter types which are not used so far. If in future other types will be needed, they have to be added. The pre- and post processing is not yet implemented. If an automate machine restart is required, the `Reboot` method of the `Win32_OperatingSystem` can be used.

Chapter 7

Conclusion

7.1 Criticism of the Windows Management Instrumentation

The Windows Management Instrumentation (WMI) is a powerful instrument for accessing and monitoring Windows resources. There exist classes for almost every manageable resource in Windows. But for the configuration of resources, WMI has to be expanded. Many classes provide none or only a few methods to manage Windows resources. Also, most of the properties are read-only.

7.2 Criticism of the Project

The Windows translator is able to configure an Ethernet card automatically. For this, a translation XSLT as well as a restriction XSLT were created, which are named `windows-xp.xsl`. Then a distributor `DistWMI.java` was implemented that distributes the data of the resulting file through WMI. Actually, it was planned to configure also a dial-up modem. But this is not possible as WMI provides no methods for this.

The distributer `DistWMI.java`, as it was implemented, can only configure resources through methods. Since many WMI classes provide no methods, it would probably be an idea to configure some resources anyway through the Windows registry. It is possible to access the registry using WMI. The `StdRegProv` class of the `root/default` namespace allows WMI scripts to interact with the Windows registry. In any case, the distributor has to be expanded to configure dial-up modems and other services that are configurable with Verinec.

Retrospective, a lot of time was invested in the research. It was difficult to find a good solution for configuring Windows machines remotely using a Java API. Also the approach chosen is not perfect. Even if it seemed as, when the documentation on WMI was read.

Fact is, that the Windows translator still needs a lot of investigation before it works as it should. There are some services that can not be handled so far, as there are no methods to manage them. Furthermore, there are services that are not studied in this thesis and consequently have to be analyzed in the future.

Bibliography

[1] David Buchmann: *Verinec Translation Module*, University of Fribourg, Switzerland

[2] D. Jungo, D. Buchmann, U. Ultes-Nitsche: *The Role of Simulation in a Network Configuration Engineering Approach*, Department of Computer Science, University of Fribourg, Switzerland

[3] Peter Monadjemi, 2002: *Windows XP Home Edition: Konfiguration, Kommunikation, Profitipps*, Mark + Technik Verlag, ISBN: 3-8272-6138-4

[4] jRegistryKey
http://www.bayequities.com/tech/Products/jreg_key.shtml
(last visited: 17.06.2005)

[5] JNIRegistry
http://www.trustice.com/java/jnireg (last visited: 17.06.2005)

[6] Microsoft's .NET Homepage
http://www.microsoft.com/net/ (last visited: 17.06.2005)

[7] Dinar Dalvi [et al.], 2001: *Professional XML for .NET Developers*, Wrox Press, ISBN: 1-861005-31-8

[8] Download WMI for Windows 95/98/NT
http://www.microsoft.com/downloads (last visited: 17.06.2005)

[9] WBEM Services
http://wbemservices.sourceforge.net/ (last visited: 17.06.2005)

[10] Jawin - A Java/Win32 interoperability project
http://jawinproject.sourceforge.net. (last visited: 17.06.2005)

[11] Holger Schwichtenberg, 2003: *Windows Scripting: Automatisierte Systemadiministration mit VBScript, Visual Basic 6.0 und Visual Basic .NET unter COM und dem .NET Framework*, Addison-Wesley, ISBN: 3-8273-2061-5

[12] G. Stemp, D. Tsaltas, B. Wells, 2002/2003: *WMI Scripting Primer: Part 1, 2 and 3*, Microsoft Corporation
Part 1: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnclinic/html/scripting06112002.asp (last visited: 17.06.2005)
Part 2: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnclinic/html/scripting08132002.asp (last visited: 17.06.2005)
Part 3: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnclinic/html/scripting01142003.asp (last visited: 17.06.2005)

[13] DMTF Common Information Model (CIM) Standards
http://www.dmtf.org/standards/cim/ (last visited: 17.06.2005)

[14] Windows Management Instrumentation Tutorial
http://www.microsoft.com/downloads/details.aspx?familyid=720F0CAE-64A7-457F-BB95-E4F33E0CBC55&displaylang=en (last visited: 17.06.2005)

[15] WMI Qualifiers
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_qualifiers.asp (last visited: 17.06.2005)

[16] WMI Scripting API
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/scripting_api_for_wmi.asp (last visited: 17.06.2005)

[17] WMI Reference
http://msdn.microsoft.com/library/default.asp?url=/library/en-us/wmisdk/wmi/wmi_reference.asp (last visited: 17.06.2005)

[18] Robert Elsenpeter, Toby J. Velte, 2002: *Windows XP Professional Network Administration*, McGraw-Hill/Osborne, ISBN: 0-07-222504-1

[19] Microsoft TechNet Script Center
http://www.microsoft.com/technet/scriptcenter/default.mspx
(last visited: 17.06.2005)

Appendix A

Acronyms

API Application Programming Interface: *An interface through which an application software communicates and exchange data with other software.*

CIM Common Information Model: *An implementation-neutral, object-oriented schema for describing network management information.*

CIMOM Common Information Model Object Manager: *Facilitates the interaction between consumer and provider whereas all requests from the WMI pass through the CIMOM.*

CMIP Common Management Information Protocol: *A protocol for network management, which defines the communication between network management applications and network agents.*

COM Component Object Model: *A Microsoft technology for software componentry, which is used to enable cross-software communication.*

DCOM Distributed Component Object Model: *A Microsoft proprietary technology for software components distributed over several networked computers.*

DHCP Dynamic Host Configuration Protocol: *A client-server networking protocol for automatically configuring networked computers.*

DLL Dynamic Link Library: *A Microsoft Windows binary application library file format.*

DMTF Desktop Management Task Force: *An industry consortium that develops, supports, and maintains standards for systems management of PC systems and products, to reduce total cost of ownership.*

GUI Graphical User Interface: *A method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text.*

IPX Internetwork Packet Exchange: *A network protocol developed by Novell.*

JDOM Java Document Object Model: *A library for working with XML completely based on Java concepts.*

JNI Java Native Interface: *A programming framework that allows Java code running in the Java virtual machine (VM) to call and be called by native applications (programs specific to a hardware and operating system platform) and libraries written in other languages.*

NIC Network Interface Card: *A piece of computer hardware designed to provide for computer communication over a computer network.*

POTS Plain Old Telephone Service: *Services available from analogue telephones prior to the introduction of electronic telephone exchanges into the public switched telephone network.*

SNMP Simple Network Management Protocol: *A protocol to administrate computers and network components remotely.*

SQL Structured Query Language: *A computer language used to create, modify and retrieve data from relational database management systems.*

WBEM Web-Based Enterprise Management: *An initiative based on a set of management and Internet standard technologies developed to unify the management of enterprise computing environments.*

WMI Windows Management Instrumentation: *Allows scripting languages like VBScript to manage Windows PCs and servers, both locally and remotely.*

WQL WMI Query Language: *A subset of the Structured Query Language (SQL) with minor semantic changes to support WMI.*

XML Extensible Markup Language: *A W3C-recommended general-purpose markup language for creating special-purpose markup languages (it is a metaformat), used to facilitate the sharing of structured text and information across the Internet.*

XSLT eXtensible Stylesheet Language Transformations: *An XML-based scripting language used for transforming XML documents into other XML or plain-text documents.*

Appendix B

Result-WMI Schema

This is part of the configuration.xsd schema.

```
<!-- ******** Windows Management Instrumentation WMI ******** -->

  <xs:element name="result-wmi">
    <xs:annotation>
      <xs:documentation>
        An XML with the configuration data to be distributed
        through the Windows Management Instrumentation (WMI).
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="wmi" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name="wmi">
    <xs:annotation>
      <xs:documentation>
        Contains the neccessary data to access the wmi class.
      </xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
       <xs:element ref="method" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="class" type="xs:string" use="required">
        <xs:annotation>
          <xs:documentation>
            The name of the wmi class used to handle the service.
          </xs:documentation>
        </xs:annotation>
      </xs:attribute>
      <xs:attribute name="namspace" type="xs:string" use="required">
```

```
      <xs:annotation>
        <xs:documentation>
          The namespace where the wmi class is stored.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="wqlKey" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>
          Defines the name of the property that forms the basis of
          the WQL WHERE clause.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="wqlKeyValue" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>
          Defines the appropriate value to wqlKey.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="method">
  <xs:annotation>
    <xs:documentation>
      Defines the method to be executed.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
    <xs:sequence>
     <xs:element ref="param" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>
          The name of the method to be executed.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="param">
  <xs:annotation>
   <xs:documentation>
     The params passed to the method.
   </xs:documentation>
  </xs:annotation>
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="param-value" minOccurs="1" maxOccurs="unbounded">
        <xs:annotation>
          <xs:documentation>
            If there is more than one param-value, they are passed
            as ArrayOfString to WMI.
          </xs:documentation>
        </xs:annotation>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="name" type="xs:string" use="required">
      <xs:annotation>
        <xs:documentation>
          The name of the input parameter assigned to the method.
        </xs:documentation>
      </xs:annotation>
    </xs:attribute>
    <xs:attribute name="type" type="xs:string" use="required">
      <xs:annotation>
       <xs:documentation>
         The type of the input parameter assigned to the method.
         So far only ArrayOfString is supported.
       </xs:documentation>
      </xs:annotation>
    </xs:attribute>
  </xs:complexType>
</xs:element>

<xs:element name="param-value">
  <xs:annotation>
    <xs:documentation>
      The value of the input parameter.
    </xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:attribute name="value" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Appendix C

Directory Organization on the CD

```
+-WindowsTranslator
  |
  |
  |+-doc
  |  |
  |  |+-experiments
  |  |
  |  |+-presentations
  |  |
  |  |+-report
  |     |
  |     |+-latex
  |     |   |
  |     |   |+-fig
  |     |
  |     |+-pdf
  |
  |
  |+-java
  |  |
  |  |+-doc
  |  |  |
  |  |  |+-api
  |  |
  |  |+-jars
  |  |
  |  |+-res
  |  |  |
  |  |  |+-translation
  |  |     |
  |  |     |+-ethernet
  |  |        |
  |  |        |+-restrictors
  |  |        |
  |  |        |+-translators
  |  |
```

```
|  |+-src
|  |  |
|  |  |+-verinec
|  |     |
|  |     |+-translation
|  |        |
|  |        |+-doc-files
|  |        |
|  |        |+-gui
|  |        |
|  |        |+-repository
|  |
|  |+-test
|     |
|     |+-verinec
|        |
|        |+-translation
|           |
|           |+-repository
|
|
|+-lib
```