

ISSN 0280-5316
ISRN LUTFD2/TFRT--5655--SE

Supervision of computer equipment In ABB OperateIT using WMI

Jens Axelsson

Department of Automatic Control
Lund Institute of Technology
October 2000

Department of Automatic Control Lund Institute of Technology Box 118 SE-221 00 Lund Sweden	<i>Document name</i> MASTER THESIS	
	<i>Date of issue</i> October 2000	
	<i>Document Number</i> ISRN LUTFD2/TFRT—5655--SE	
<i>Author(s)</i> Jens Axelsson	<i>Supervisor</i> Jan Gjerseth ABB Karl-Erik Årzén LTH	
	<i>Sponsoring organization</i>	
<i>Title and subtitle</i> Supervision of computer equipment in ABB operateIT using WMI (Övervakning av datorutrustning i ABB OperateIT med hjälp av WMI)		
<i>Abstract</i> This document details the investigation and implementation of a product intended for supervision and management of standard computer and office equipment. The product is run from inside the ABB Operate ^{IT} Platform, as an Aspect System. The Aspect receives its data through communication with the CIM Object Manger, as suggested in the WBEM Initiative. The document also details the WBEM Initiative and some the standards brought forward by this initiative.		
<i>Keywords</i>		
<i>Classification system and/or index terms (if any)</i>		
<i>Supplementary bibliographical information</i>		
<i>ISSN and key title</i> 0280-5316		<i>ISBN</i>
<i>Language</i> English	<i>Number of pages</i> 66	<i>Recipient's notes</i>
<i>Security classification</i>		

The report may be ordered from the Department of Automatic Control or borrowed through:
University Library 2, Box 3, SE-221 00 Lund, Sweden
Fax +46 46 222 44 22 E-mail ub2@ub2.se

Master Theses at: Lund Institute of Technology
Author: Jens Axelsson
Examiner: Karl-Erik Årzén, Associate Professor in Automatic Control
Supervisor: Jan Gjerseth
Site: ABB Automation Products AB
Document number: 3BSE022901/0.6

Operate^{IT}

WMI Powered Aspect

Abstract

This document details the investigation and implementation of a product intended for supervision and management of standard computer and office equipment. The product is run from inside the ABB Operate^{IT} Platform, as an Aspect System. The Aspect receives its data through communication with the CIM Object Manger, as suggested in the WBEM Initiative. The document also details the WBEM Initiative and some the standards brought forward by this initiative.

Table of Contents

1. PROJECT SPECIFICATION	7
1.1 MOTIVATION	7
1.2 OBJECTIVES	7
2. FEASIBILITY-STUDY	9
2.1 DIRECT ACCESS TO THE DRIVER	9
2.2 ACCESSING SPECIALLY DESIGNED MANAGEMENT APPLICATIONS	10
2.3 UTILIZING WINDOWS MANAGEMENT INSTRUMENTATION	11
2.4 CONCLUSION FROM THE FEASIBILITY-STUDY	11
3. DESCRIPTION OF OPERATE^{IT}	14
3.1 INTENDED USE AND POSSIBILITIES	14
3.2 GOALS FOR THE OPERATE ^{IT} PLATFORM	15
3.3 THE ABB ASPECT OBJECT MODEL	15
3.3.1 ASPECT OBJECTS TM	16
3.3.2 ASPECTS	17
3.4 SYSTEM STATUS FRAMEWORK	17
3.4.1 SYSTEM STATUS PROVIDER	18
3.4.2 SYSTEM STATUS VIEWER	18
4. PRE-STUDY OF WINDOWS MANAGEMENT INSTRUMENTATION.....	21
4.1 WEB-BASED ENTERPRISE MANAGEMENT	21
4.1.1 DISTRIBUTED MANAGEMENT TASK FORCE	21
4.1.2 COMMON INFORMATION MODEL	22
4.1.3 OTHER DMTF STANDARDS	24
4.1.4 THE STATUS PROPERTY	24
4.2 DESCRIPTION OF WMI	25
4.2.1 WMI QUERY LANGUAGE	26
4.3 RISK ESTIMATION	27
5. ABB ASPECT OBJECT MODEL AND CIM.....	29
5.1 REPRESENTATION OF CIM PROPERTIES IN ASPECT OBJECTS	29
5.2 CREATION OF WMI POWERED ASPECT OBJECTS	30
5.3 MAPPING	32
5.3.1 HARD CODING	32
5.3.2 MAPPING THE STATUS PROPERTY	33
5.3.3 GENERIC MAPPING	34
6. PROTOTYPING THE WMI ASPECT	38
6.1 SOFTWARE ENGINEERING	38
6.1.1 REQUIREMENTS ENGINEERING	39
6.1.2 SOFTWARE PROTOTYPING	39

6.2	OUTLINE DEFINITION	41
6.3	THE IMPLEMENTATION	41
6.3.1	CREATING THE SKELETON OF THE ASPECT	41
6.3.2	CONNECTING TO THE CIM OBJECT MANAGER	42
6.3.3	SELECTING WMI PROPERTIES	43
6.3.4	NOTIFICATION OF CHANGE IN WMI PROPERTIES	44
6.3.5	DESIGN OF THE CONNECTION TO WMI	45
7.	IMPLEMENTATION OF A SIMPLE WMI PROVIDER.....	47
7.1	PROVIDER	47
7.2	MSDN TUTORIAL	47
7.3	ABBGENERATOR PROVIDER	48
7.3.1	FUNCTIONAL DESCRIPTION	48
7.3.2	IMPLEMENTATION	48
8.	EVALUATION OF THE WMI CLIENT-PROVIDER SOLUTION	51
8.1	WMI PROVIDER	51
8.2	WMI CLIENT	52
8.3	WMI CLIENT-PROVIDER	53
8.4	CONCLUDING REMARKS	54
9.	FUTURE WORK.....	55
9.1	IMPROVED WMI ASPECT	55
9.2	WMI RELATED EXTENSIONS	56
10.	CONCLUSION	58
10.1	FULLFILLMENT OF THE OBJECTIVES	58
10.2	CONCLUDING REMARKS	58
	APPENDIX A – DESCRIPTION OF ABBREVIATIONS AND WORDS	60
	APPENDIX B – MOF FILE FOR THE WMI PROVIDER ABBGENERATOR.....	62
	APPENDIX C – WMI ASPECT USER GUIDE.....	64
C.1	USER INTERFACE COMPONENTS	64
C.2	TUTORIAL	67
C.3	PITFALL AVOIDANCE	68
	REFERENCES.....	69

Document outline

All abbreviations used in the text, except ones that has become names, will be described, or at least typed out in full, in Appendix A – Description of Abbreviations and Words.

1. Project Specification

This chapter answers the questions: Why is this project initiated and what are the objectives of this investigation?

2. Feasibility-Study

This chapter describes a selection of possible methods to get the Operate^{IT} Platform, an application that is run on the Microsoft Windows 2000 operation system, to supervise and manage standard computer and office equipment.

3. Description of Operate^{IT}

This chapter describes the Operate^{IT} Platform, why it is used, possibilities with it, its goals, and some features of the product. The last part of this section is devoted to the system status framework.

4. Pre-Study of Windows Management Instrumentation

This chapter summarizes a pre-study of WMI and the results will be used to increase the understanding of the problem domain. The purpose of the pre-study is to lie as a foundation for the outline of the WMI Aspect prototype.

This chapter starts off with a description of the roots of WMI that can be found in WBEM. The information content herein are from press releases and other material found on the DMTF's website. The pre-study will then continue with a more detailed look upon WMI. The chapter will then conclude with a number of estimated risks that come with relying on WMI for critical and non-critical tasks.

5. ABB Aspect Object Model and CIM

This chapter will cover design specific details on how to access management data published by WMI from the Operate^{IT} Workplace. This will be done through a WMI powered Aspect that the engineer can use as a building block when making structures of managed elements with Aspect Objects.

There are three sections in the chapter, the first describes how the two object models can be fitted together, i.e. CIM and the Aspect Object Model. The second section covers the building of structures and how the Aspects should be configured. The last section describes how the mapping of WMI Properties to OPC Properties could be done.

6. Prototyping the WMI Aspect

This chapter covers the prototyping of the Operate^{IT} WMI Aspect. The first section, 6.1, will be a motivation of the development model chosen, namely prototyping. The main source of information contained in the motivation is, Sommerville, I. (1995) [9]. Then the next section is a brief overview of the tools used to build the foundation of the aspect

system. The last section, 6.3, covers implementation specific details, like how the connection to the CIM Object Manager is established and how the communication works.

7. Implementation of a Simple WMI Provider

This chapter covers WMI Provider writing in general and specially the implementation details of a sample provider, the ABBGenerator.

8. Evaluation of the WMI Client-Provider Solution

This chapter describes the functionality of the WMI Client-Provider implemented in chapter 6 and 7 of this document. This description will be more focused on the functionality of the WMI Provider and the WMI Client from a users point of view, where as chapter 6 and 7 focuses more on how they are implemented. How the pieces of this chapter fits together can be seen in Figure 2-2, but the box at the bottom “Managed System” does not exist.

9. Future Work

This chapter is divided into two parts. Part one covers the evolution of the WMI Powered Aspect, and part two covers how WMI might be used in future products.

10. Conclusion

This chapter summarizes the final conclusions drawn from the entire lead-time of the project.

1. Project Specification

This chapter answers the questions: Why is this project initiated and what are the objectives of this investigation?

1.1 Motivation

The Operate^{IT} workstation is fundamentally designed to supervise and manage automated plants with, among other things, sensors, controllers, and feeders. For more information about Operate^{IT} see chapter 3, "Description of Operate^{IT}". Sensors, controllers, and feeders are all specially designed for use in the industry and are therefore often rather expensive. Equipment constructed for use in the industry are, compared to office equipment, more expensive mainly because (1) higher quality, (2) smaller series, and (3) less competition. An alternative to using industry standard products is the use of *common of the shelf*, COTS, components for non-critical tasks. As more COTS components make their way into the world of automation the necessity to supervise them grow. Since COTS components are not usually designed for the often dirty and trying environment found in most factories, the fault frequency will be even greater than with the more costly industry standard equipment. The two factors above put together give that the need to detect faults in standard office equipment will increase over the time to come.

Naturally ABB Automation Products AB, the developers of Operate^{IT}, does not want to spend a lot of time and effort on designing special functionality for fault detection in standard office equipment. It is also not likely that the constructors of the equipment are keen on developing specially designed drivers were data could easily be published and processed in Operate^{IT}.

1.2 Objectives

In this section the main objectives and additional objectives are presented. These objectives are presented in natural language and do not contain any kind of unique identifier for later reference. The objectives are also not presented in any given order of importance.

- 1) This project opts to make it easy to supervise and manage standard office and computer equipment using the existing ABB Operator Station, Operate^{IT}.
- 2) Examine WBEM and describe Microsoft's implementation, WMI.
- 3) Investigate how the CIM-schema can be extended with ABB Objects.
- 4) Investigate if it is possible, and what is required, to make Operate^{IT} specific data available to third-party software via WMI with the help of a WMI Provider.
- 5) Examine the support WMI is getting from third-party developers.
- 6) Investigate how to map the CIM into the ABB Aspect Object Model.

2. Feasibility-Study

This chapter describes a selection of possible methods to get the Operate^{IT} Platform, an application that is run on the Microsoft Windows 2000 operation system, to supervise and manage standard computer and office equipment.

The most obvious solutions are presented below:

- 1) Direct access to the driver
- 2) Accessing specially designed management applications
- 3) Utilizing Windows Management Instrumentation, WMI

Possibilities and constraints put upon the method originate in Windows NT's layered structure, and the Intel processor architecture, not particularly in the Operate^{IT} Platform. If the Operate^{IT} Platform were run on Microsoft DOS none of the solutions mentioned above would be possible. The solution in that case would most likely be to use the `port` command, reading and writing to different ports accessing the hardware directly.

2.1 Direct access to the driver

In the first solution there is a hard coupling to the device, even though not as hard coupling as using the `port` command mentioned above. This is how communication is traditionally done with devices in Windows NT/2000.

Windows NT/2000 is based on a layered structure, as can be seen in Figure 2-1 [2], where the standard user applications runs in user mode whereas the kernel runs in kernel mode. These are two of the four, 0 to 3, protection modes [1] introduced with the Intel 80386 processor, and give the running code different privileges. The kernel runs in mode 0, most privileged, and the user applications in processor mode 3, least privileged. This protection mechanism provides the ability to limit access to certain segments or pages based on the privilege levels. A user mode process that wants access to the hardware has to make a system call to the kernel. The processor will then enter mode 0 and the request will be sent to the appropriate module in the kernel, where it will be processed and then the call will be passed to the hardware through one of the three exit points, hardware device drivers, hardware abstraction layer, or graphic device driver. Any answer or result has to be propagated back the same way.

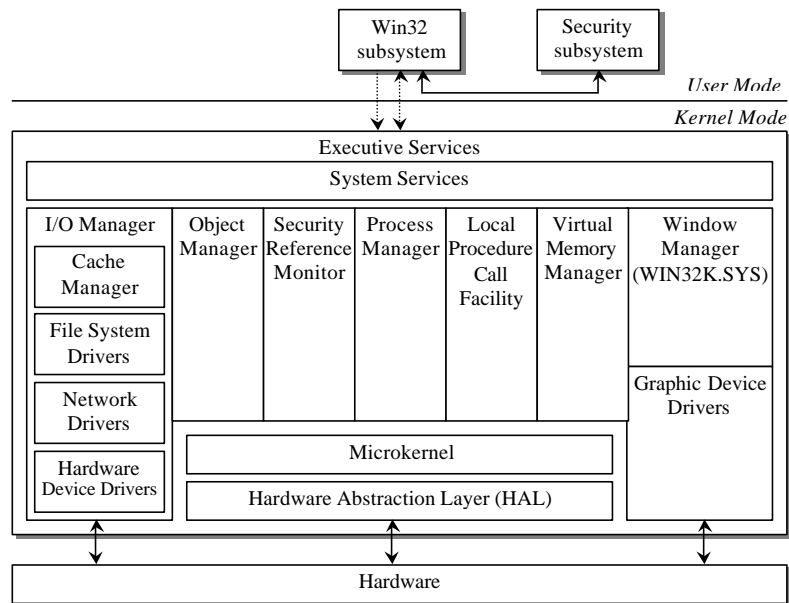


Figure 2-1. Windows NT 4.0 Modular Architecture.

This solution will require intimate knowledge of the supervised equipment, but might result in the most flexible design of the management aspect, and will also result in the shortest access time to the managed device. The fundamental difference between this solution and the other solutions mentioned is that in those solutions more software layers are added on top, as applications or services running in user mode.

This method could be recommended only when a small and well-defined set of equipment is to be supervised. The equipment in question should also be well known to the developer, this since all communication is done directly with the device.

2.2 Accessing specially designed management applications

Some hardware manufacturers develop applications, specially designed for the supervision and management of their particular product. This is the case with for example Hewlett-Packard. They develop a tool called "HP Web JetAdmin" [3]. The tool is designed for configuration and supervision of printers, network scanners, and CD-ROMs over a local network or even Internet. What this program does is very close to the objectives of this project, but it should be done from within Operate^{IT}. If it would be possible to access the application from Operate^{IT} in a seamless manner this could be a viable solution. Another problem that comes to mind is that for example JetAdmin only supervise a small set of the standard equipment that could be found on the market, namely HP products and printers that are Standard Printer MIB compliant. This gives that to increase the set of equipment that Operate^{IT} is able to manage, more management applications has to be accessed and when such an application is not

available, no supervision is possible. One can also imagine that if the applications are at all accessible the functionality and how to access it might vary greatly.

2.3 Utilizing Windows Management Instrumentation

Windows Management Instrumentation, WMI, is Microsoft's implementation of WBEM a standard for distributed device management. One of the most attractive objectives of WBEM is that all management data is supposed to be accessed in a uniform way. The implementation of WMI consists of two strongly coupled software layers, see Figure 2-2, but in between the device driver and the management application. The top-layer is a service developed by Microsoft. To that service so called WMI Providers are attached at run-time when needed. The device driver developer and hardware manufacturer develop these WMI Providers. This leads to both the advantages and the drawbacks with WMI.

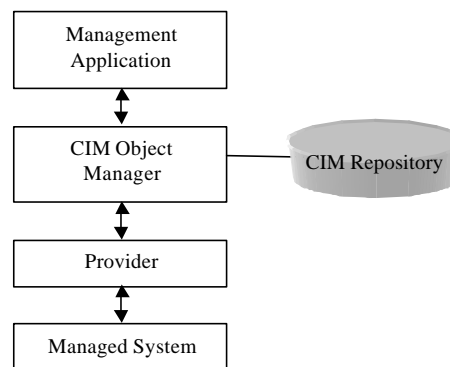


Figure 2-2. WMI Architecture.

The main advantages are that the management data is retrieved from the device by a component developed by the persons that one could expect to have the best knowledge about the device, and that it is then accessed by the management application in a uniform way.

The main disadvantages are that the standard is rather new, and the implementation is even newer. The full implementation is, as of today, only available in Windows 2000. This gives that the standard might change and might not get the strong support expected. This can lead to few fully-fledged implementations, by third party developers, of WMI Providers. Another disadvantage, even though it might be insignificant in most situations, is that adding more software layers will require more CPU power, this will inevitably make the final management product slower.

2.4 Conclusion from the Feasibility-Study

The first solution, direct driver access, can be used in special cases where a low-latency high-speed management data access is more important than a short development time. This solution will not be described in greater detail in this report. Even with the

disadvantages mentioned above, in section 2.3, WMI is the only viable solution for a general and generic management application. Because of the uniform management data access, once a WMI powered management application is developed, adding new devices requires no additional development.

3. Description of Operate^{IT}

This chapter describes the Operate^{IT} Platform, why it is used, possibilities with it, its goals, and some features of the product. The last part of this section is devoted to the system status framework.

3.1 Intended Use and Possibilities

The possibility to see what happens on the factory floor is severely limited due to for example spatial distribution, and machine safety shielding. Still the need to know what happens is great, due to the fact that malfunction might pose a threat to the machines, the staff, the public, or at least the production. This knowledge could be used for error detection, maintenance, and process optimization. Desired is also to move the information out of the often noisy and not seldom ill conditioned environment of the factory floor.

There are a number of issues with plant supervision, like should one use centralized or distributed management, and what information should be presented to different users. The points of view put forward above, are all in favor of centralized supervision, i.e. all the information from all different apparatus in the plant is brought to one location. But this is not the best solution in all situations [7], when maintenance personal for example are correcting malfunction in a machine the need to diagnose the machine might occur frequent. If the distance from the machine to the supervision room is great this could be tedious work. This problem could be solved in a number of ways:

- 1) Phone conversation between the maintenance personnel and the personnel in the supervision room
- 2) Redistribute the information around the plant floor
- 3) Use the faulty machines own interface
- 4) Use WAP enabled hardware to distribute the information

Solution 1 might be the natural and cheapest way but might also be error prone, due to difficulties inherited with vocal communication. The often noisy environment found in factories makes vocal communication even harder. Solution 2 might be a good thing if the cost of the terminals is low. Solution 3 is a traditional solution but requires the personnel to understand and cope with a number of different implementations. Solution 4, the use of rather new technology as of 2000, here this is can be that information is brought to a wireless handheld device, like a Palm Pilot or a WAP enabled mobile phone. All solutions are possible with Operate^{IT}, due to the high degree of scalability. The WAP solution was demonstrated on the International Trade Fair Interkama '99 in Düsseldorf, using the WAP enabled mobile phone, Ericsson R380s, as of that time a prototype.

The second issue mentioned above is, what information should be presented to different users. In today's systems the users spend a lot of time searching for the information instead of using the information. This is because the information is scattered in different binders, databases, files, and around the web. The problem is not lack of information, rather the lack of structure in the information. One of the main concepts of the Operate^{IT} workplace is to focus on the information. The information should only be one click away in a context sensitive manner. This means that depending on who clicks and where, the information the person is confronted with will be customized accordingly. The focus on information distribution is to ease the navigation and thereby optimize the productivity of the coworkers of the plant. The concept also includes focusing on the right information and to not present unnecessary and unwanted information and thus clutter the view. An example could be that administrative personnel, would in most cases not want to see the blueprints of an entity, but this might be the first choice for maintenance personnel. Further, both administrative and maintenance personnel might want historic data. But the administrative personnel as an excel sheet for further processing and presentation, and the maintenance personnel as an overview graph to look for trends to decide about preventative maintenance.

The main concepts to deal with these two issues mentioned above are ABB Aspect ObjectsTM, this via scalability and flexibility. This concept is described in more detail in the following sections below.

3.2 Goals for the Operate^{IT} Platform

A transcript of the four major goals of the Operate^{IT} Platform, as presented in [8], are

- 1) to provide a system that solves the customer's problem, is easily understood and allows him to work efficiently with all the complexities inherent in an automation system,
- 2) to ensure (or at least make possible) that the software developed for the different functions can work together as one consistent and integrated system,
- 3) to provide a system in which reusable solutions can be developed. As ABB does much of its business by providing customers with solutions it is important that the solutions can be reused, and
- 4) that the architecture and the platform that supports this architecture will make it possible to develop software that is compliant with the architecture at low and predictable cost.

3.3 The ABB Aspect Object Model

The ABB Aspect Object Model addresses the issue of presenting information and allowing a user to operate on information in a consistent way. The model also addresses how different functions are integrated into the system in a way natural to the user. The ABB Aspects Object Model is based on ABB Aspect Objects and Aspects.

3.3.1 Aspect Objects™

The fundamental building block in Operate™ is the Aspect Object. Aspect Objects tend to map to real world entities, like a valve, a motor, a controller, or a sensor. But can also map to abstract entities like user groups, functions, or software. Aspect Objects can be put in different hierarchical structures for different viewing purposes. Examples of structures can be seen in Figure 3-1. One Aspect Object can be found in many structures. These structures can be used to navigate, and easily find objects of interest. When for example the “V37, Block Valve” in Figure 3-1a, Functional Structure, is found the user can switch to the Location Structure to find where in the plant the Block Valve is located. Then to identify to what network the block valve is connected the user only needs to switch to the Control Structure. This is one example of how information is easily cycled through for commonly performed tasks.

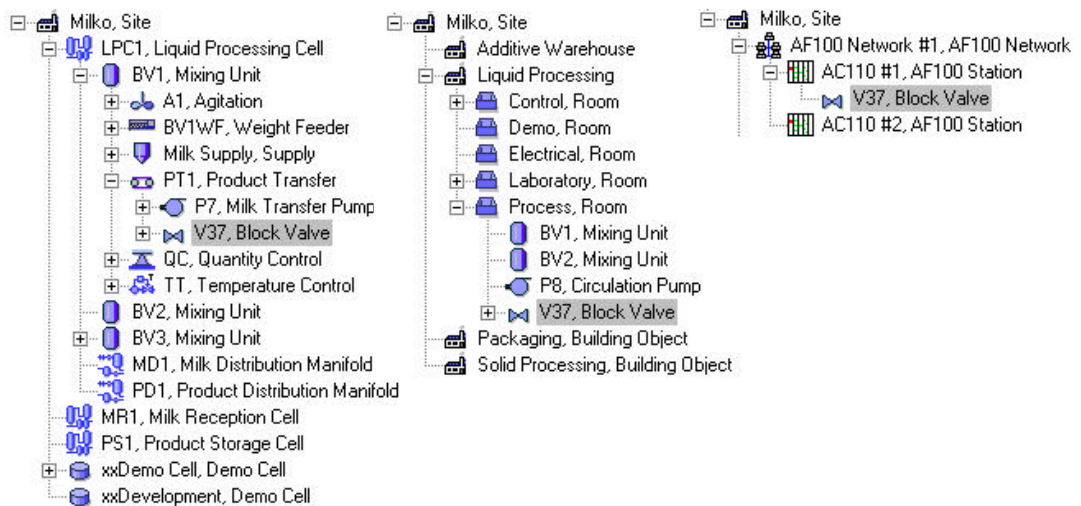


Figure 3-1. Three possible structures, the same entity of a block valve are marked in gray. From the left a) Functional Structure, b) Location Structure, and c) Control Structure.

One benefit of the Aspect Objects are the possibility to re-use solutions. This is an actual advantage because a lot of objects in the real world consist of the same type of objects. This is especially true in automated plants, where for example multiple identical tank farms might exist for cooling and/or buffering. The engineer can create a type solution consisting of many Aspect Objects like for example a Supply, a Transfer Pump and a Block Valve. The engineer can then select the aspects of his liking to be contained in the objects.

When a type solution is created it is possible to instantiate any number of copies of it, this to easily create any number of complex instances. This reduces the time consumed by creating many similar objects; this time can then be invested in assuring that the type solution created is correct and well optimized. Every time this type solution then is re-used, the engineer then saves time and can be sure that new instances are well

designed. The use of type objects also leads to another advantage that if one wants to update the objects, one only needs to change the type object and this change is then reflected in all the instantiated objects.

3.3.2 Aspects

The Aspect Objects are a more conceptual part of the ABB Aspect Object Model, the real strength is the Aspects of the objects. The Aspect Objects can be thought of as folders, i.e. they function as containers that contain the information carrier. Aspects can be seen as different ways to view the objects, i.e. viewpoints. The different viewpoints are suitable for different purposes and persons of different professions. Examples of possible aspects on an Aspect Object are name, functional description, operator graphics, maintenance record, and so forth. These are all representations of the same real world entity but from different perspectives. The aspects not of any interest to the operator are hidden; this so that the amount of information will be kept to a minimum because unnecessary information will just clutter the view. One of the ideas is that the user is supposed to, from any aspect, be able to navigate to any other aspect of the Aspect Object. This, together with the possibility to navigate the structures of Aspect Objects, does make it easy to find specific information and to get an overview of the entire plant.

All interaction with the Aspect Objects is done through its Aspects, i.e. methods and properties only exist on the aspects. A user can only perceive an Aspect Object through its aspects as opposed to the systems internal representation of the Aspect Object as a GUID. This gives that multiple Aspect Objects can have for example name aspects with identical name attributes. It is then still different objects but the user will not be able to tell them apart.

The Aspect Directory is the component that keeps track of and stores the association between aspects and Aspect Objects. Aspects, as Aspect Objects, are identified by a GUID. This construction makes it possible to move them between systems, it does in this process retain its GUID. If for example an Aspect Object is developed in one system, then exported to another system, and then modified in the first system, and then re-exported to the second system it will be correctly updated in that system.

3.4 System Status Framework

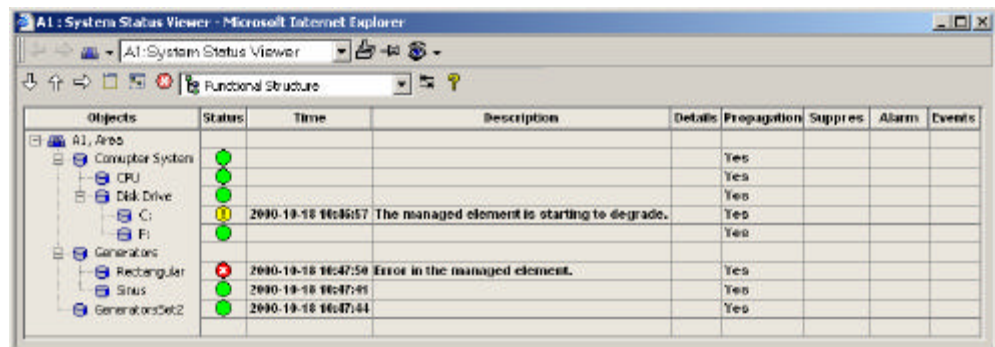
The functionality of system status framework is described in the document [13]. The system status framework consists of two complementing parts: the viewer and provider requirements. The viewer is an Aspect that is used to view the status of Aspect Objects providing system status in accordance with the requirements put upon them.

3.4.1 System Status Provider

Any aspect can be a system status provider by just fulfilling the requirements put upon them. But if an Aspect Object has more than one Aspect that functions as a system status provider the result is undefined. The requirements put upon a system status provider are (1) implementing the `IAfwSystemStatusReporter` interface, (2) must at least provide the subscribable OPC Property `S_STATUS`, and (3) detailed description should be implemented as aspect views and have the aspect key `System Status Details`. The key, in requirement three, is added through Operate^{IT} Workplace when the system is configured at run-time. The other two requirements are fulfilled at design-time. To requirement one can be added that the OPC Properties `S_TIME` and `S_DESCRIPTION` ought to also be supported for additional information about any error presented [14].

3.4.2 System Status Viewer

The System Status Viewer is used to view the status of Aspect Objects with an Aspect fulfilling the requirements put upon a System Status Provider. The System Status Viewer implements an aspect system that displays an overview of the status for the Aspect Objects in the structure. This can be seen in Figure 3-2. At the top of the figure are buttons and boxes for navigating structures in Operate^{IT}, below this are buttons and boxes to control and navigate the structure presented by the System Status Viewer.



The screenshot shows a web browser window titled "A1: System Status Viewer - Microsoft Internet Explorer". The address bar shows "A1: System Status Viewer". Below the browser window is a table with the following columns: Objects, Status, Time, Description, Details, Propagation, Suppres, Alarm, and Events. The table contains the following data:

Objects	Status	Time	Description	Details	Propagation	Suppres	Alarm	Events
A1, Area								
Computer System	●				Yes			
CPU	●				Yes			
Disk Drive	●				Yes			
C:	⚠	2010-10-18 16:46:57	The managed element is starting to degrade.		Yes			
F:	●				Yes			
Generators								
Rectangular	⚠	2010-10-18 16:47:50	Error in the managed element.		Yes			
Sinus	●	2010-10-18 16:47:49			Yes			
GeneratorsSet2	●	2010-10-18 16:47:44			Yes			

Figure 3-2. System Status Viewer.

The main area are divided into a number of fields of which the rightmost are the structure selected showing Aspects Objects complying to the requirements of a System Status Provider. The next field shows the status of the managed element represented by the Aspect Object. In this case all objects are 'Ok' except the logical disk 'C:' and the signal generator generating a rectangular wave. These are in the state 'Warning' respectively 'Error'. The states are represented by: a green circle for 'Ok', a white cross on a red background for 'Error', and an exclamation sign on yellow background for

'Warning'. The next field shows the time of the last change in status for the Aspect Object. The field after that is a description of current state, not needed for the 'Ok' state.

In the scenario presented in Figure 3-2 the disk drive is ok but the 'c: partition' is in a warning state, how can this be? One possibility is that the disk drive is working fine but the 'c: partition' is almost full, and accordingly the status is set to 'Warning'. Another possibility is that the file structure on the 'c: partition' is corrupt.



4. Pre-Study of Windows Management Instrumentation

This chapter summarizes a pre-study of WMI and the results will be used to increase the understanding of the problem domain. The purpose of the pre-study is to lie as a foundation for the outline of the WMI Aspect prototype.

This chapter starts off with a description of the roots of WMI that can be found in WBEM. The information content herein are from press releases and other material found on the DMTF's website. The pre-study will then continue with a more detailed look upon WMI. WMI is Microsoft's implementation of WBEM and is an integral part of Windows 2000. It is strongly connected with the Microsoft Management Console, and the Windows Driver Model. WMI is a technology pushed by Microsoft with the release of Windows 2000. The chapter will then conclude with a number of estimated risks that come with relying on WMI for critical and non-critical tasks.

4.1 Web-Based Enterprise Management

Web-Based Enterprise Management, WBEM, started as an industry initiative in 1996 and has evolved into Distributed Management Task Force in 1999 [4]. The problem WBEM tried to address is that element management is done in isolation, i.e. little or no integration, and that the burden of interpreting objects are placed on each management application. There are some standard protocols for management but they only address a small piece each, like for example network devices utilizes SNMP, desktop systems DMI, telecom devices CMIP and yet other devices uses their own private protocols.

The goal of WBEM was (1) to harness the power of the web for management interoperation, (2) build a Common Information Model for management, and (3) to integrate existing standards (SNMP, DMI, CMIP, etc). With "harness the power of the web" it is meant that management data and the actual managed elements should be accessible from the Internet, and the second vision states that all managed entities are accessed through a single unified standard protocol in this case xmlCIM.

4.1.1 Distributed Management Task Force

The Distributed Management Task Force, DMTF, is an industry organization that is leading the development, adoption and unification of management standards and initiatives for desktop, enterprise and Internet environments. The DMTF is chartered to adopt, create, and maintain the specifications and technologies that provide management tools with the ability to discover, deploy, and control management data in a standard way. Working with key technology vendors and affiliated standard groups, the DMTF is enabling a more integrated, cost effective, and less crisis-driven approach to management through interoperable management solutions. Worth noting is that the meaning of the D in DMTF has altered, the former meaning was Desktop but it has changed to Distributed.

The initial WBEM initiative that came from BMC Software, Cisco Systems, Inc., Compaq, Intel, and Microsoft has transformed into DMTF with the current board members shown in Table 4-1. Together with these 13 board members, there are other types of members, Contributing Members, Associate Members, Alliance Partner Members, Customer Advisory Board Members, and Academic Alliance Members. There are about 200 members of the DMTF of which a handful are Ericsson, AT&T, Fujitsu Limited, Motorola, Nokia, and Lund Institute of Technology.

Board Members
3Com
Avaya
Cisco
Compaq Computer Corp.
Dell Computer Corp.
Hewlett-Packard Company
IBM/Tivoli Systems, Inc.
Intel Corporation
Microsoft Corporation
NEC Corporation
Novell
Sun Microsystems, Inc.
Symantec Corporation

Table 4-1. Table showing current members of the DMTF board.

Different management applications need to have a common understanding of the managed elements regardless of how that information is stored or transported, DMTF's solution to this problem is called Common Information Model and is based on an object-oriented model, in that sense that it incorporate for example such ideas as classes, instances, inheritance, properties, and methods. CIM is described in more detail in the following section, section 4.1.2.

4.1.2 Common Information Model

One of the main objectives of the DMTF was to establish a uniform standard on how management information should be represented; this standard is called Common Information Model or CIM for short. The management model is divided into the following conceptual layers [5]:

- 1) Core Model – an information model that captures notations applicable to all domains of management.
- 2) Common Models – information models that capture notions common to particular management domains but independent of a particular technology or implementation. The common domains include Systems, Applications, Devices, Users, Networks, Policies and Databases.

- 3) Extension Models – represent technology-specific extensions of the Common Models. These models are specific to environments, such as operating systems, or to vendors.

CIM is based on concepts known from object-orientated design, and is in the documents from DMTF modeled in UML. The basic building block in the CIM is the class, in UML modeled as a box. The attributes of a class are known as properties, the classes are furthermore built of methods. The methods can be used to interact with the managed element; one example could be a reset method to reset a malfunctioning logical device. The CIM object hierarchy is based on inheritance and the top of the CIM core model v2.4 can be seen in Figure 4-1. Inheritance is a subclass/superclass relationship where the subclass inherits the properties and methods from the superclass. The CIM does not support multiple inheritances.

As described in chapter 3, “Description of Operate^{IT}”, objects and aspects are all thrown into a void uniquely identified only by their GUID. This is called an object identity model. This in contrast to the keyed object model used in CIM where all class instances are uniquely named and referenced by the class’ keys. Instances of CIM classes are uniquely identified by their key properties, their class name, and a namespace identifier. As CIM is only an information model the implementation may require different identifiers, one may use GUIDs another like DEN may use distinguished names.

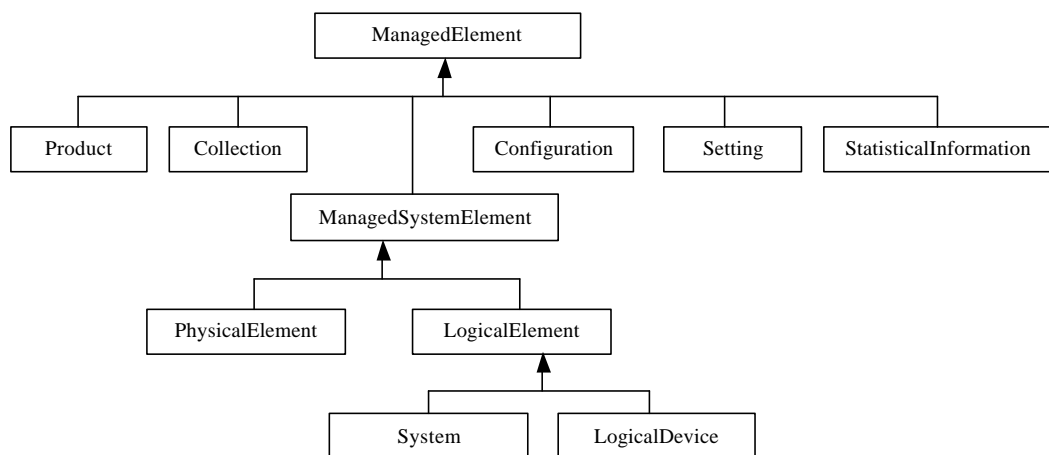


Figure 4-1. The CIM object hierarchy v2.4.

The Managed Element class roots the CIM object hierarchy and acts as a reference for associations that apply to all entities in the hierarchy. Managed System Elements represent Systems, components of Systems, any kinds of services (functionality), software and networks. The definition of “System” in the CIM context is quite broad, ranging from computer systems and dedicated devices, to application systems and network domains. Both Logical and Physical Elements are subclasses of Managed

System Element. Further definition and specification of these subclasses are provided in the Core and Common Models. For example, System and Logical Device objects are subclasses of Logical Element, defined in the Core Model.

The Common model is an information model that captures notions that are common to particular management areas, but independent of a particular technology or implementation. The information model is specific enough to provide a basis for the development of management applications. This model provides a set of base classes for extension into the area of technology-specific schemas. The Core and Common models together are expressed as the CIM schema. The currently defined common areas are Systems, Devices, Applications, Networks, and Physical.

The CIM Schemas are described by a text format called Managed Object Format, MOF, and its syntax reminds a lot of the syntax of C++. The CIM Schemas can also be expressed in XML as defined by the "XML Mapping White Paper" [6].

4.1.3 Other DMTF Standards

As mentioned above WBEM is not supposed to eliminate other standards for management but rather incorporate them and, by that, the work put into them. Here two DMTF Standards incorporated in CIM will be briefly described, this for completeness and for possible future extensions of this investigation. The two standards are DMI and DEN.

Desktop Management Interface, DMI, details a standard framework for managing networked desktop systems and servers and details a standard way of sending DMI management information across a network. DMI also incorporate a highly flexible event model, where it is possible to describe and filter the events. To see how CIM and DMI are related to each other look at Figure 4-2.

Directory-Enabled Networks, DEN, is an initiative to develop a standard and extensible directory schema foundation for heterogeneous networks providing interoperable directory services for networking. The DEN specification allows applications to transparently leverage network infrastructure from the users perspective, and support distributed network-wide service creation, provisioning and management. DEN specifies a common data model, CIM v2.3, with LDAP mappings from CIM to X.500.

4.1.4 The Status Property

The classes that apply to many real world office objects like Printers, Hard Drives, Memory, CPU, and so forth are all classes that inherit from the class `CIM_ManagedSystemElement`. Its position in the CIM Schema v2.4 can be seen in Figure 4-1 above. `CIM_ManagedSystemElement` represent systems (computer, network, storage library, and application system), the software running on them, the functionality provided by them, and the hardware that compose them. This class has the properties `Name`, `Description`, `Caption`, `Install Date`, and

`Status`. The `Status` property is of great importance in this project since it describes operational and non-operational states for a managed system element. Like all other properties, the `Status` property is inherited by all derived classes. This gives that for example Printers, Hard Drives, and Memory, and so forth all have their own copy of the same type of property.

The status property is a 10-character string that, in CIM Schema v2.0, can have the values shown in Table 4.2. In CIM Core Model v2.4 white paper [10] the explanation to why the status property remained a short string instead of a uint16, i.e. an enumeration, when moving to version 2 of the CIM is that implementations built on version 1 of the CIM relied on the string format.

OK
Error
Degraded
Unknown
Pred Fail
Starting
Stopping
Service

Table 4-2. The defined values of the `Status` property.

The states of the status property is described in the Microsoft Platform SDK documentation as the current status of the object. The operational states are “OK”, “Degraded”, and “Pred Fail”. “Pred Fail”, i.e. an element where failure is predicted in the near future. It is a state set by for example an element such as a SMART-enabled hard drive or other elements designed to foresee a coming failure. All the other states are non-operational states.

The unknown state should be read as that the provider is not able to gather the management information. This could for example be the case because of network failure and such types of events. In version 2.4 of the CIM Schema two more detailed unknown states are added, “No Contact”, i.e. the entity is known to exist but no management contact has been made, and “Lost Comm”. These two states add some information to why the status is unknown.

4.2 Description of WMI

WMI is currently based on the DMTF CIM Schema v2.0, with a substantial addition in form of extended schemas, the Win32 extended schema. One example of how Win32 extends the CIM Schema is that `Win32_Printer` inherits from `CIM_Printer`, and adds operating system specific properties like `Attributes`, e.g. if it's the default printer, `DriverName`, and `PrintProcessor`.

On computers running Microsoft Windows NT/2000, WMI runs as a service. On computers running Windows 95/98, the Windows Management Service runs as an application. The Windows Management Service is invoked on either operating system by running the executable file WinMgmt.exe. It is possible to stop and restart WMI manually whenever one wants to. In Figure 4-2 all that is contained inside the dotted oval is the functions of the WinMgmt service, providers are COM components linked to the process when needed.

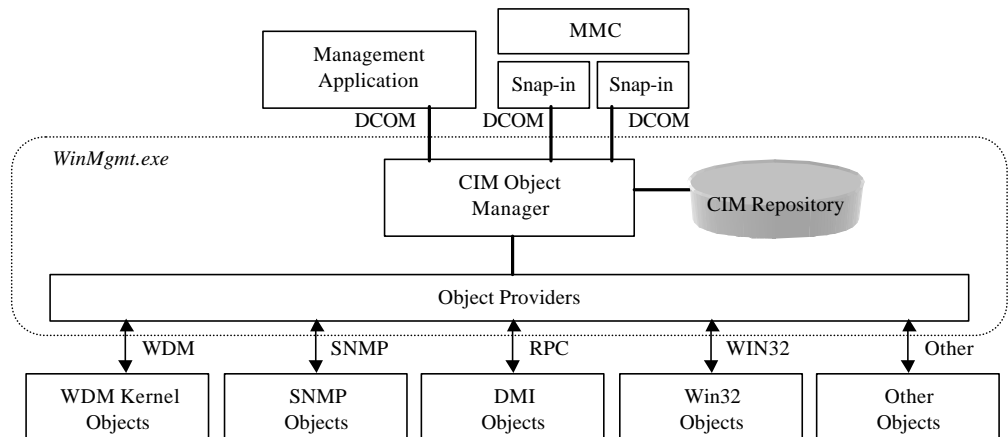


Figure 4-2. The WMI Architecture in detail.

A central goal of WBEM is uniform representation of data, and this data is encapsulated in object-oriented fashion in the CIM Objects Repository. The CIM Object Manager is a collection and management point for managed objects stored in the repository. Data is not accessed by the CIM Object Manager directly. This is done by WMI Providers whom gather information from a resource and then make it available to the CIM Object Manager. WMI Providers are described in more detail in section 7.1.

4.2.1 WMI Query Language

To access management information managed by the CIM Object Manager queries are used. The syntax of the queries is defined by the WMI Query Language, WQL. WQL is a subset of ANSI SQL. Unlike SQL, WMI can only be used to retrieve data; it cannot be used in operations involving modification, insertion, or deletion of data. To access management information, queries are used, examples of such queries are:

```
“SELECT * FROM Win32_LogicalDisk WHERE DeviceID='C:'”
```

This query will retrieve all properties from the logical disk named 'C:'. This query can be specialized to retrieve only a subset of the properties of a class:

```
“SELECT FreeSpace,Size FROM Win32_LogicalDisk WHERE DeviceID='C:'”
```

This query will only retrieve the amount of free space and the size of the logical disk named 'C:', to further specialize the query, more conditions can be added:

```
“SELECT FreeSpace FROM Win32_LogicalDisk WHERE
DeviceID='C:' AND FreeSpace < 10485760”
```

In this last example a second condition is added to the query. The property, FreeSpace, will only be retrieved if the conditions are fulfilled, i.e. there is less than 10Mb of free space on the logical disk with the device identifier 'C:'. Of course version two and three of the query are not pure optimizations of query one in that sense that they do exactly the same just better they are just examples of how to use some features found in WQL. They are optimizations in that sense that if one wants to do what is done in example three, using query one and then stripping the result will be a waste of resources.

4.3 Risk Estimation

As stated in the feasibility-study in section 2.3 a risk could be seen that the support for WMI is as of yet not complete. But after some further investigation a number of implementations using the CIM have been found.

A number of large companies have announced support for the CIM Standard in their Management Software. In Table 4-3 some of these products can be seen along with its respective developer.

Management Software	Developer
Insite Manager	Compaq
IT Assistant	Dell
OpenView Network Node Manager	Hewlett-Packard
SMS	Microsoft
ManageWise	Novell
NetView	IBM/Tivoli
Unicenter TNG	Computer Associates

Table 4-3. Management Software with support for the CIM Standard.

The major intention with the uniform management information model, CIM, is for different applications from different application vendors to access the management information in a uniform way. But another advantage with the uniform information model is the possibility to use it even when the management application and the managed device are developed by the same company. One of the advantages with using CIM in this situation, are the elimination of having to design a special purpose information model just for the companies own needs.

The broad support WBEM, and thus WMI, is getting from the industry with applications from major vendors already supporting the CIM Standard it is more than likely that the risk of using WMI for supervision and management is small.



5. ABB Aspect Object Model and CIM

This chapter will cover design specific details on how to access management data published by WMI from the Operate^{IT} Workplace. This will be done through a WMI Powered Aspect that the engineer can use as a building block when making structures of managed elements with Aspect Objects.

There are three sections in the chapter, the first describes how the two object models can be fitted together, i.e. CIM and the Aspect Object Model. The second section covers the building of structures and how the Aspects should be configured. The last section describes how the mapping of WMI Properties to OPC Properties could be done.

5.1 Representation of CIM Properties in Aspect Objects

CIM is an object model and so is the ABB Aspect Object Model so there is a strong possibility that they could be fitted together. There is a number of potential ways to fit the models, all with different advantages and drawbacks. The two main objectives are to make the final solution consistent with the ABB Aspect Object Model and also easy to comprehend and work with. Some of the possible solutions are shown in Figure 5-1, Figure 5-2, and Figure 5-3 below.

The first possibility is to map one WMI Object to one Aspect Object. This case is shown in Figure 5-1. One downside to this model is that multiple types of configurations must be included in the same aspect if it should be possible to import more than the status property from the WMI object. This can easily be solved in a tidy way with for example a tab control that has one tab for each imported WMI property. This will be discussed in more detail later in the document. Another issue with this model is the question, how this complies with object-oriented design, since this solution actually has multiple aspects (different WMI Properties) mapped to one single Operate^{IT} Aspect.

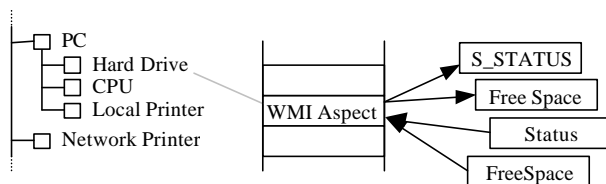


Figure 5-1. Multiple objects with single WMI aspect with multiple properties.

The second solution is to map one Operate^{IT} Aspect to one WMI Property, as can be seen in Figure 5-2. The downsides to the first model are here eliminated, there is one type of configuration view for each Operate^{IT} Aspect and an Operate^{IT} Aspect is closely related to what is meant by an aspect in object-orientation. Unfortunately other downsides has emerged, one being that the number of aspects on an Aspect Object will increase tremendously. This might lead to that the view of the aspect directory is

cluttered, and that it might be a more tedious work creating multiple aspects, depending on the level of automation.

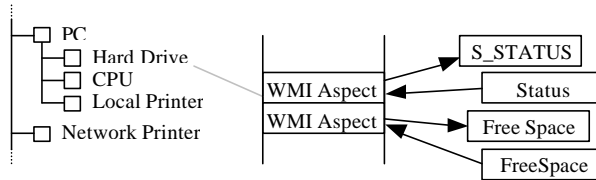


Figure 5-2. Multiple objects with multiple WMI aspect with single properties.

The third way that is presented here is to have one Aspect Object mapping to a subset of the whole WMI database or namespace in WMI terminology. What this would look like is modeled in Figure 5-3. In Figure 5-3 multiple aspects are used to model different aspects of an office, it is also possible to have a single Operate^{IT} Aspect for the whole WMI namespace. This model does not work well with the aggregation used in the Operate^{IT} Class structure where for example status codes are propagated up in the hierarchy. This is mainly due to the fact that status codes are bound to objects. This leads to that the status of the office equipment object in Figure 5-3 could either be in the state 'Ok', 'Error', or 'Warning' and all the details of the problem would have to be shown in the system status description. This is mainly a problem when multiple warnings and errors exist at the same time.

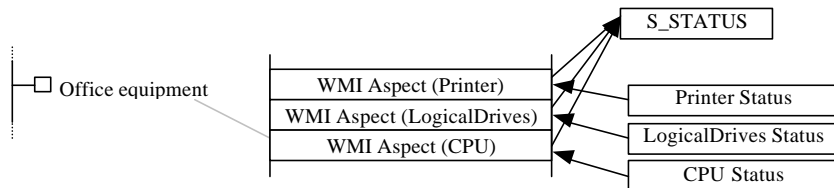


Figure 5-3. Single object with multiple aspects.

Of these three possible ways of including WMI objects in Operate^{IT}, the third is the most flexible in that meaning that it is possible to derive the first and the second solution out of it, and the second is the least flexible. As a matter of fact the design choice here is if one should impose a structure upon the user like in solution one or give the user total freedom like in solution three.

5.2 Creation of WMI Powered Aspect Objects

Not all computers are created equal, some have one processor and another has two some have one hard drive and others have multiple drives with multiple partitions. There are also for example plug-and-play enabled USB Devices that the user expects to show up, or at least be easily accessible when plugged-in to the system.

Three levels of update automation can be identified:

- 1) Manual Creation,

- 2) Up-Loader, and
- 3) Auto Update Service.

The least complicated solution is to fulfill the first level. This solution completely lacks automation. The user has to create the Aspect Object in the structures of choice and add a WMI Aspect to the object. The WMI Aspect then has to be configured to reflect the functionality that was intended with the Aspect Object. If the systems setup changes, like for example a disk is repartitioned, the configuration of the aspects has to be updated and new Aspect Objects might have to be created. This will or won't be a problem depending on the amount of work needed to configure the WMI Aspect. If for example only the WMI Status property is mapped to Operate^{IT}'s System Status, see next section 5.3 Mapping, no configuration is needed, then the problem will be minimal and thus the benefit of a higher level of automation will be small.

The second level is to have some sort of up-loader aspect that, after a minimal amount of configuration, can fill the structures, for example location structure and/or control structure, with Aspect Objects already configured with the right information. This up-loader should for example add two WMI Powered CPU Aspect Objects to the structure if there are two CPUs in the computer system examined. The configuration of this aspect could, let the engineer identify the entity to up-load, what structures it should be added to, and what aggregated entities to check for. An example of what the configuration view described above might look like can be seen in Figure 5-4. This solution makes the construction and configuration substantially easier for the user, this since the user needs not know what devices are connected to the computer system. But on the other hand it would require a lot of engineering and forethought by the constructor of the implementation.

Figure 5-4. Possible configuration view for WMI Aspect Objects Up-Loader.

If the third level is fulfilled Aspect Objects representing the device will directly be created in the appropriate structures when devices are connected. This would require a lot of further investigation but would be a nice feature and ease the use for operators. To make this direct update possible a process would have to run all the time and constantly monitor, for example, the WMI events “__InstanceCreationEvent”, and “__InstanceDeletionEvent” updating any structures when changed. This

solution is fully automated and requires minimal interaction with the user, and guarantees that true and updated information is accessible at all times.

What level should be implemented? Is level two and three obtainable? The second question will be left for future investigation, and the answer to the first is that it depends on the outcome of the first question. The second question could also be broadened to include how much work has to be done to reach level two and three. But for a prototype, level one should be sufficient, even though it lacks the additional eye-candy.

5.3 Mapping

What mapping means here is that a number of WMI properties should get a meaningful sense in the Operate^{IT} Platform. Example of this is to map the WMI Status property to S_STATUS in Operate^{IT}. But other more advanced functions could also be thought of, like for example when the size of free space on the hard drive is below 10 Mb, S_STATUS should be set to warning. This requires a more advanced form of mapping.

Three types of mapping will be discussed here, when the mapping is done in the source code of the aspect, when the mapping is done in a simple way at runtime, or a generic mapping.

5.3.1 Hard Coding

The easiest way of mapping, from the creator's point of view, is where the configuration is done mainly or totally in the source-code of the aspect. It is easy to use and create aspects in this manner but very inflexible, and the use of each aspect is severely limited.

This method could be used mainly in three cases, namely

- 1) Prototyping,
- 2) Limited support, and
- 3) Advanced queries.

To prototype a feature is a good second step after a feasibility study. This prototype could be used to show the possibilities of the functionality, but it will lack a lot of user interaction functionality, and the possibility for the user to customize it. This will make the prototype only useful for demonstration purposes, not for evolving the functionality. This since changing something will require the code to be recompiled, so this will restrict the user testing and evaluating the prototype.

The second situation where a hard coded version could be used is if the support for the feature should be limited. If for example only status for hard drives and CPU utilization should be present then the most cost effective method could be to hard code the aspects instead of creating a fancy GUI to configure the aspect.

The third situation where one might consider using a hard coded aspects is where the queries or the answers are too advanced to create a user interface that the user can comprehend. In this situation one would use it mainly to make the life of the users easier.

5.3.2 Mapping the Status Property

If the users interest only lie in the status of the office equipment it is sufficient to map the WMI status property to Operate^{IT} Platforms System Status properties, of which required are, S_STATUS, S_DESCRIPTION, and S_TIME. This functionality could easily be configured by the user, with some minor assistance from the platform. The configuration view would only consist of for example a combo box showing a number of instances of some predefined classes.

If for example Win32_Keyboard, Win32_PointingDevice, and Win32_LogicalDisk are selected then the combo box might show "Enhanced (101- or 102-key)", "Microsoft PS/2 Mouse", "A:" and "C:" to the user to choose from. To build a hard drive object the user then creates a new object, includes the WMI aspect and then chooses to configure the aspect then selects the "C:" from the combo box and clicks apply.

The mapping of WMI Status to Operate^{IT}'s S_STATUS could be configurable by the user, but a natural way to map is shown in Table 5-1. The Unknown state is, as can be read in section 4.1.4, a state where communication failure is detected and the managed element might function perfectly. But it might still be a good idea to signal Warning, thus making the operator aware of the fact that the communication with the element is lost. The warning signal should be accomplished with a detailed description of the situation so the operator can start to look for the faulty component in the right place.

WMI Status	S_STATUS
OK	Ok
Error	Error
Degraded	Warning
Unknown	Ok/Warning
Pred Fail	Warning
Starting	Out of Service
Stopping	Out of Service
Service	Out of Service

Table 5-1. The WMI Status property and what it could mean in Operate^{IT}.

If one estimate that there will be too many objects in the combo box showing instances, this could be if one has selected 10 to 20 interesting classes and for example there are multiple hard drives with multiple partitions. An easy solution could be to have two combo boxes, one showing classes and the other instances of that class. This would

effectively eliminate clutter and the data could more naturally be presented in alphabetical order.

This would give the user an easy to use but rather inflexible instrument to supervise the condition of office equipment. If one wants a more flexible instrument the mapping must be done in a generic way.

5.3.3 Generic Mapping

The user might want to monitor the free space of the hard drive and issue a warning when it goes above 10% of the total hard drive space, to be able to do this a more advanced type of query must be used. This might lead to a paradox, where a better and more flexible tool is less used because it becomes too complex.

In this system complexity might not have to be a problem because it is possible to create object types with pre-configured aspects. Object types with pre-configured aspects are Aspect Objects that are setup with any number of ready-configured Aspects before the product is shipped. This is done by the developers or engineers. When the Aspect Objects are instantiated in a structure the only choice the user has to make is whether the aspects are to be duplicated or linked to the Aspect Object. Linking are preferred because it enables the engineer to do a change at one place and have multiple instances changed. This concept leads to that one can make a rather complex and thus difficult to use interface and still have a tool that is easy to use by the general user. This tool would then be used in two ways, (1) just importing ready aspects and, (2) constructing aspects to map WMI properties to OPC properties in Operate^{IT}. An example of a generic user interface is presented in Figure 5-5. The figure shows two combo boxes at the top and two list boxes, and below them a textbox.

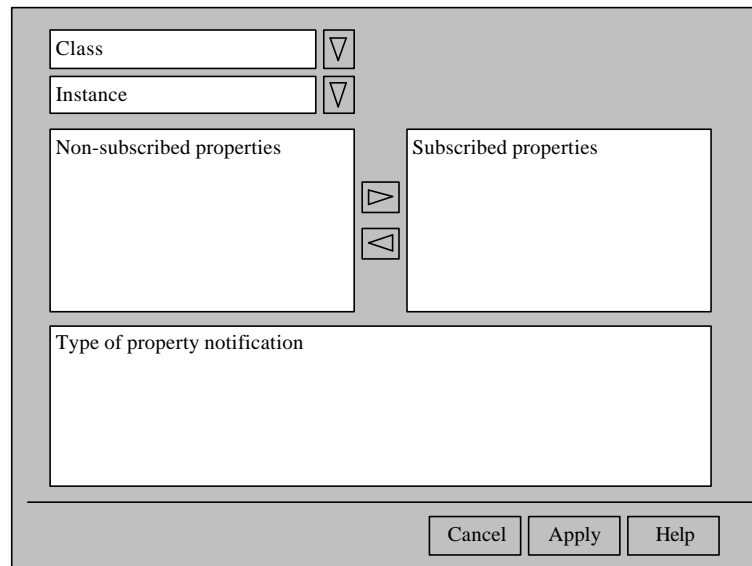


Figure 5-5. An example of a generic interface.

The first combo box labeled “Class” is initially the only non-disabled object and it shows all, or a subset, of the WMI classes. When one class is selected the combo box is disabled and the other objects are enabled. The user then selects an instance from the combo box labeled “Instance”, the properties of that instantiated class is then shown in the list box “Non-subscribed properties”. The user can now subscribe to any properties of his liking.

The steps before were all simple; the more advanced step is where one chooses how these properties will be mapped to OPC properties on the Operate^{IT} Platform. This mapping is done in the area at the bottom of the view. The user should in the ideal situation not be limited to map one WMI property to one OPC property. The user should be able to map one WMI property to multiple OPC properties and vice versa. This could be used as shown in the following example of a hard drive object.

The system status property, S_STATUS, should reflect changes in the WMI status property of the hard drive, and an additional status property called FreeSpace should report the WMI property FreeSpace, so far all is simple but the user would also like to have a condition on the WIM property FreeSpace. If there is less than 10 MB free space left, and the S_STATUS is in another state than error, it should be set to warning and if there is less then 5 MB free space left S_STATUS should be set to error.

This last statement of the example leads to that the WMI property FreeSpace should not only affect the OPC property FreeSpace. It also has to be connected in a conditional way to S_STATUS.

There are a number of methods to access WMI properties; of these two will be presented here,

- 1) `IWbemServices::GetObject` followed by
`IWbemClassObject::Get`, or
- 2) `IWbemServices::ExecNotificationQueryAsync`.

In the first method one asks for an instance of a WMI class and then on that instance asks for the specific property that is returned as a data type VARIANT. This is the upper path in Figure 5-6. This could be used to access static properties or polling constantly changing properties. A static property is for example the size of a hard drive, or a name of an instance. Properties of this sort could for example be displayed in a process pictures and for other information purposes. Constantly changing property can for example be `Win32_Processor:LoadPercentage`, this type of properties one might want to show in a trend diagram. `LoadPercentage` for example could be used for debugging and optimization purposes.

In the second method, schematically described in the lower path in Figure 5-6, one starts by setting up a callback sink that will perform some action, this together with an advanced WQL query is passed to the method. Nothing is returned until the condition in the query is satisfied.

The WQL query can be of the following form:

```
“SELECT * FROM __InstanceModificationEvent WITHIN <poll
time> WHERE TargetInstance ISA <WMI Class> AND
TargetInstance.DeviceID=<WMI Instance> AND
TargetInstance.<WMI property> <condition>”
```

As for example:

```
“SELECT * FROM __InstanceModificationEvent WITHIN 3
WHERE TargetInstance ISA 'Win32_LogicalDisk' AND
TargetInstance.DeviceID='F:' AND
TargetInstance.FreeSpace < 10485760”
```

The example above will call the callback sink when the logical disk 'F:' has less than 10 MB of free disk space. In the callback sink one could for example set the `S_STATUS` to warning or set an alarm. The example above will call the sink every time a write is done to the disk when it is less than 10 MB free space. This type of access could preferably be used when properties change rather infrequent.

In the last example above the first part of the query could be filled in from the combo boxes and list boxes and the last part could be typed in the box at the bottom of the example user interface shown in Figure 5-5. But the user should also be able to select between the two cases 1, and 2 described above so a more advanced list box should be used.

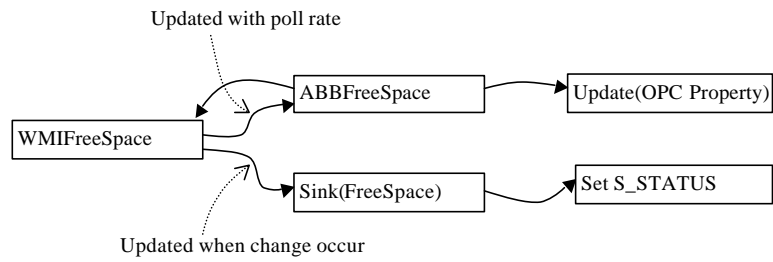


Figure 5-6. Visualization of how one WMI property is mapped to two properties in different ways.

It should also be mentioned that the sink could also update any OPC properties, a little less natural way would be to monitor a status property by polling it. So one could imagine a possible arrow from the Sink(FreeSpace) box to the Update(OPC Property) box.

6. Prototyping the WMI Aspect

This chapter covers the prototyping of the Operate^{IT} WMI Aspect. The first section, 6.1, will be a motivation of the development model chosen, namely prototyping. The main source of information contained in the motivation is, Sommerville, I. (1995) [9]. Then the next section is a brief overview of the tools used to build the foundation of the aspect system. The last section, 6.3, covers implementation specific details, like how the connection to the CIM Object Manager is established and how the communication works.

6.1 Software Engineering

There are four fundamental process activities in the software process; these four activities can then be decomposed in different ways by different software processes. The fundamental activities are Software Specification, Software Development, Software Validation, and Software Evolution. Every activity builds directly on the outcome of all the previous activities and should be executed with consideration of the future activities to ease the activities to come. Software Specification is the activity where requirements are extracted and the software is designed. This will be covered in more detail in section 6.1.1. The second step is where an implementation is done in accordance with the design specification produced earlier. Validation is where the implementation is tested to see if it functions in compliance with the functional description of the software. This includes finding defects in the software. Evolution is the final stage of the process, and presumably the longest, the software must evolve to meet the changing customer needs.

These four fundamental process activities can be subdivided and described in greater detail, there are a number of well known ways to do this. Among these are the waterfall approach, evolutionary development, formal transformation, and system assembly from reusable components. The first two are commonly used, the third is more experimental, and the fourth is on the increase mainly thanks to the object-orientation paradigm. These processes will, with the exception of formal transformation, be described below. A software process describes the stages, what to do in each and what deliverables to produce in each.

The most common and well know process is the waterfall model that is derived from other engineering processes. During each step in the waterfall model one or more documents are produced. The main concept here is that after each action is performed satisfactory the produced documents are put into baseline and if, in a later stage, a defect would be found in a baselined document, the defect is corrected and the outcome is numbered with a new version number. This gives this model high visibility, i.e. the progress of the process is externally visible. On the other hand the rapidity, i.e. how fast a system can be delivered using the process, tend to be low. The lack of rapidity is due to that a lot of work, other than only the implementation of the software,

has to be done. Evolutionary development is based on the idea of developing an initial implementation, exposing this to any stakeholders and collect their requirements and this to refine the product. This is iterated through many versions until an adequate system has been developed. The process 'system assembly from reusable components' is focused on developing the system from existing components, where only a smaller part of the implementation is spent to glue the components together. This is an attractive approach since re-use is often rather inexpensive compared to constructing new. One problem with re-use is that it is almost impossible to get exactly what one wants. Also if the documentation of the component is insufficient, later stages in the process like testing and maintenance will be more difficult, but this is a problem with all ill documented code in all processes. Microsoft is pushing this technology with their concept of Component Object Model, COM, where components even could be added at software runtime.

6.1.1 Requirements Engineering

Requirements engineering is the activity of extracting requirements from anyone that may have any influence on the system design. This is a difficult activity due to the facts that it is required that the requirement specification should be complete, correct, unambiguous, consistent, and verifiable. The activity of requirements extraction can be divided into subactivities. The first and one of the more important subactivities is domain understanding. This is the activity where the analyst develops the understanding of the application domain. Domain understanding is important to engineer adequate requirements. Next the requirements are collected and classified to resolve conflicts among the collected requirements. The requirements are then prioritized to, at implementation time, be able to see where to start implement and in lack of time, which requirements to neglect. The last step will be to validate the remaining bunch of requirements, i.e. to see if they are complete, consistent and in accordance with what stakeholders really want from the system. These activities all naturally lead to greater domain understanding. This implies that the requirements will be better engineered if the steps are iterated once more. This iterating should be done until the validation activity does give a good enough result.

6.1.2 Software Prototyping

As stated before this project aims to investigate the possibility to use and the possibilities with WMI, in contrast to developing a fully operational product. To investigate this fully and to have something for demonstration purposes some sort of software product would be needed. There were a number of design decisions that had to be taken before considering implementing the functionality. Examples of these were type of software process, implementation language, and platform for the implementation.

The benefits of developing a prototype of the function to implements are

-
- 1) Misunderstandings between software developers and users may be identified as systems are demonstrated,
 - 2) Missing user services may be detected,
 - 3) A working, albeit limited, system is available to demonstrate the functionality and usefulness to management,
 - 4) The limited system could be used for initial training of future users, and
 - 5) The prototype serves as a basis for writing the specifications for a production quality system.

Some negative aspects on prototyping are as follows

- 1) Poor visibility, i.e. the process activities does not culminate in clear results, this leads to difficulties in making the progress of the project visible,
- 2) Poor maintainability, the rapidity of the project might lead to a lack of documentation, and
- 3) Bad implementations and workarounds might find it's way into the final product.

There are two types of prototyping software processes with different goals, the two methods can be seen in Figure 6-1. In method one, evolutionary prototyping, the basic idea is to develop an initial implementation, exposing it to user comment and refining this through many stages until an adequate system is developed. The outcome of this process is a delivered system. An important difference between evolutionary development and structured development is in the validation phase. Validation is only meaningful when the software is compared to a functional description. An additional problem with evolutionary prototyping is that the initial outlined structure tends to be corrupted with the constant changes made to the prototype.

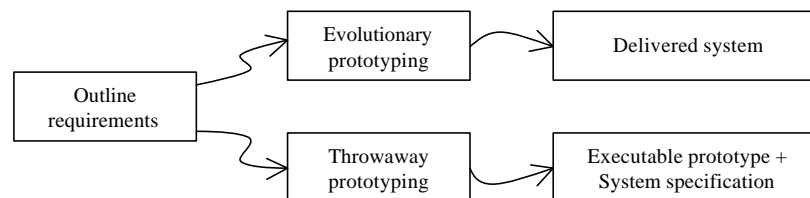


Figure 6-1. Evolutionary and throwaway prototyping, and what is delivered.

Throwaway prototyping is the other type of evolutionary development. This method is more focused on acquiring the system specification as a first step, and from this design the system. The prototype is developed and changed according to user needs, and when the prototype fits the users need it is frozen and requirements are extracted from the prototype. The final product is then based on a design made from these requirements. One advantage with this method is that the structure of the final product is as described in the system specification. This correspondence between specification and final product make the product possible to validate and easier to maintain.

6.2 Outline Definition

For the implementation the development model evolutionary prototyping was selected but the final goal of this project is not a deployable product, so the product was not iterated over until a final defect-free implementation was obtained, but rather until what can best be described as demonstrator was obtained. A demonstrator is unlike a prototype developed with evolutionary development not a product to build on but rather use for demonstration purposes, i.e. demonstrating special functionality.

As development platform Visual Studio, using Visual C++, is the natural choice because this is the platform Operate^{IT} is implemented with, and thus has the best integrated support in form of CM, from Visual SourceSafe, and automated code generation, from “ABB Automation System Application C++ Wizard”, and “AfwTools Code Generator”. Other than these, Visual Studio features no support for requirements handling or software design.

6.3 The Implementation

A vague idea existed on the design of the component but most had to evolve along with the implementation. The design was also restricted by what was generated by any automated code generators used. This section will reflect the development process in its layout. First the code skeleton was generated then the functionality was explored. When understood and refined it was inserted in a controlled fashion into the code skeleton. In this way a structure could still be kept and more functionality could be added as the domain understanding increased over the development process. At the end of this section is a description of what the design finally looked like.

6.3.1 Creating the skeleton of the Aspect

For a COM object to be able to function properly as an aspect a number of requirements have to be fulfilled, i.e. a number of interfaces have to be supported. The easiest way to fulfill these requirements is to use a wizard to generate the skeleton of the code. This is of course only possible if a wizard exists that suite ones needs. The skeleton of the Aspect was created using the “ABB Automation System Application C++ Wizard”. How this is done can be seen in document the related document [11].

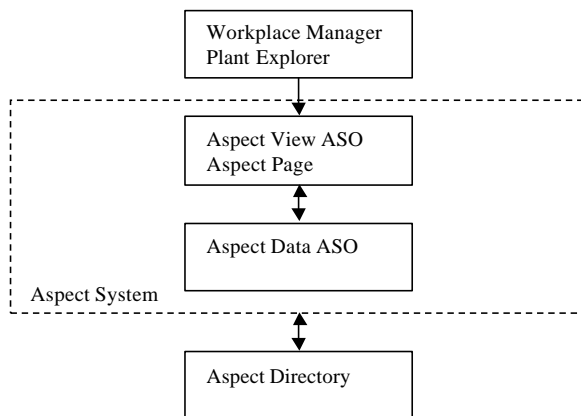


Figure 6-2. Software relationship seen from the aspects point of view.

A brief overview of the design implied by the automated code generators can be seen in Figure 6-2, which is a copy of a figure in section 1.4 of [11]. The Aspect View ASO Aspect Page is in this project a COM Object named CInfoTab and the Aspect Data ASO is basically composed of two COM Objects named CDiscStatus and CDiscStatusData but also a number of classes designed for connecting to WMI and to exchange data.

6.3.2 Connecting to the CIM Object Manager

To be able to access the functionality of WMI a connection to the functionality CIM Object Manager within the service WinMgmt.exe has to be established. As WinMgmt.exe is a service, it is running in a separate process, this gives that the address space is not the same as the one the aspect is using. This results in that communication cannot be done in the same manner as is done with in-process components. Data sent across process boundaries has to be marshaled. Microsoft has thought of this and has defined a standard for COM objects residing in different processes, called Distributed COM, or DCOM. This standard will be used when communication with the WinMgmt.exe process.

To get the interface handle to the IWbemLocator, CoCreateInstance is called as can be seen in Listing 6-1. This function returns the interface handle as the last argument. It is in this code snippet assumed that a call to the function CoInitialize(NULL), that has to be done once before connecting to COM Objects, is already done for this process. This can safely be assumed because CDiskStatus, the class that instantiates CWMIConn, is also a COM Object. The connection to WinMgmt.exe is then done with a call to the command IWbemLocator::ConnectServer.

```

CWMIConn::CWMIConn(CDiscStatus* pPar)
{
    IWbemLocator* pLoc = NULL;
  
```

```

DWORD dwRes = CoCreateInstance(CLSID_WbemLocator, 0,
    CLSCTX_INPROC_SERVER, IID_IWbemLocator, (LPVOID *) &pLoc);
CoInitializeSecurity(NULL, -1, NULL, NULL, RPC_C_AUTHN_LEVEL_NONE,
    RPC_C_IMP_LEVEL_IDENTIFY, NULL, EOAC_NONE, 0);
BSTR PathName = SysAllocString(L"\\\\.\\ROOT\\CIMV2");
HRESULT hRes = pLoc->ConnectServer(PathName, NULL, NULL,
    0, 0, 0, 0, &m_pSvc);
SysFreeString(PathName);
if (pLoc) pLoc->Release();
return;
}

```

Listing 6-1. The Connection Class' Constructor connects to WinMgmt.exe.

When later on trying to use this connection to setup event notifications with the function `IWbemServices::ExecNotificationQueryAsync` no query would be setup and an error code would be returned, namely `WBEM_E_ACCESS_DENIED`. This error can be avoided if the correct authentication information is set when accessing the proxy stub used when communicating using RPCs as with DCOM. The correct authentication information is applied using the command `CoSetProxyBlanket` as can be seen in Listing 6-2. This function call is inserted after `m_pSvc` is set using `IWbemLocator::ConnectServer`, and applied to `m_pSvc`.

```

hRes = CoSetProxyBlanket(m_pSvc, // proxy
    RPC_C_AUTHN_WINNT, // authentication service
    RPC_C_AUTHZ_NONE // authorization service
    NULL, // server principle name
    RPC_C_AUTHN_LEVEL_CALL, // authentication level
    RPC_C_IMP_LEVEL_IMPERSONATE, // impersonation level
    NULL, // identity of the client
    EOAC_NONE); // capability flags

```

Listing 6-2. The function `CoSetProxyBlanket` used to set the authentication information.

When the connection to WinMgmt.exe is done then communication with the managed elements through the CIM Object Manager can be done as described in section 6.3.3, and section 6.3.4, both found below. When no more communication is to be performed then all that is needed to disconnect is to release the reference to the interface `IWbemServices`.

6.3.3 Selecting WMI Properties

For the user to be able to select which WMI Properties to subscribe to, the properties should be shown to the user in for example a list box. The user might also be interested in what instances of managed elements are available in the system. Those should also be shown to the user, this time preferably in a combo box. What this initially was intended to look like can be seen in Figure 5-5. However one problem arose, if the user changed what class to configure, the whole or parts of the configuration would have to be invalidated. This due to the fact that what properties exist are dependent of the class.

Therefore a button was added to instantiate the WMI Class. When this is done the instances that are present on the system are enumerated in a combo box, and the combo box with the WMI Classes are locked so no change can be made to it. If the button is clicked one more time the configuration is invalidated and the combo box showing the WMI Classes is unlocked.

Listing 6-3 shows, with appropriate simplifications, how a combo box is filled with available instances of the selected WMI Class. By creating the query as is done in Listing 6-3, only querying for two properties, the number of times the inner loop is traversed is minimized.

```
IEnumWbemClassObject* pEnum = NULL;
IWbemClassObject*     pInstance = NULL;
query = L"SELECT Caption,__RELPATH FROM <Class Name>";
m_pSvc->ExecQuery(..., query, ..., &pEnum);
while (pEnum->Next(..., &pInstance, ...))
{
    pInstance->BeginEnumeration(NULL);
    while (pInstance->Next(..., &propName, &pVal, ...))
    {
        ... //add the value of the Caption propertie to combo box
    }
    if (pInstance) pInstance->Release();
}
if (pEnum) pEnum->Release();
```

Listing 6-3. Enumerating available Instances of WMI Classes.

Entering of the properties into the list box is done in more or less the same way. The three differences are that (1) the query is changed to retrieve all properties not just Caption, and __RELPATH, (2) the outer while loop are removed since all instances have the same properties, and (3) the actual name of the property is added to a list box.

The splitting of the functionality to two separate loops is done for simplicity and optimization purposes. It would be possible to query for all properties and then have an if-statement checking if it was the first instance, then adding the names of the properties to the list box, but this would result in that all the properties were retrieved for all the instances.

6.3.4 Notification of change in WMI Properties

As mentioned above `IWbemServices::ExecNotificationQueryAsync`, details can be found in section 5.3.3, would be used to let WinMgmt.exe tell the WMI Aspect when the value of WMI Properties did change. This would ease the burden on Operate^{IT} and let WinMgmt.exe do all the work. Because not all management providers support event notification it is possible to set a poll-rate for the query. This will tell WinMgmt.exe to check the property at the poll-rate, compare it to the last time it was pulled if a change in the property has occurred, the rest of the conditions in the query are checked if they also are true a notification is sent to the process making the query.

The last argument to `ExecNotificationQueryAsync` is a pointer to the callback sink. The callback sink is an instantiated COM Object, with at least the functions `Indicate()`, and `SetStatus()`, which are abstract functions in the sinks super class, `IWbemObjectSink`, inherited by the sink. When `WinMgmt.exe` recognize that the conditions of a query is fulfilled it makes a RPC to the `Indicate()` function in the sink object passed to it.

6.3.5 Design of the Connection to WMI

When the functions “connecting to the CIM Object Manager” and “queering for management data in an asynchronous fashion” was explored and refined, two classes were designed, `CWMIConn`, and `CMySinc` that was supposed to function as an abstraction layer between WMI and the Aspects main function. How the two abstraction classes, the main function of the aspect, and WMI fits together can be seen in Figure 6-3. An arrow with dashed line denotes construction, and thus a call to the class' constructor. An arrow with a solid line denotes a function call. If an arrow starts in the gray area with the class name it means that the call is done from the class constructor. A description of the steps performed to connect and setup the callback sink can be found below the figure.

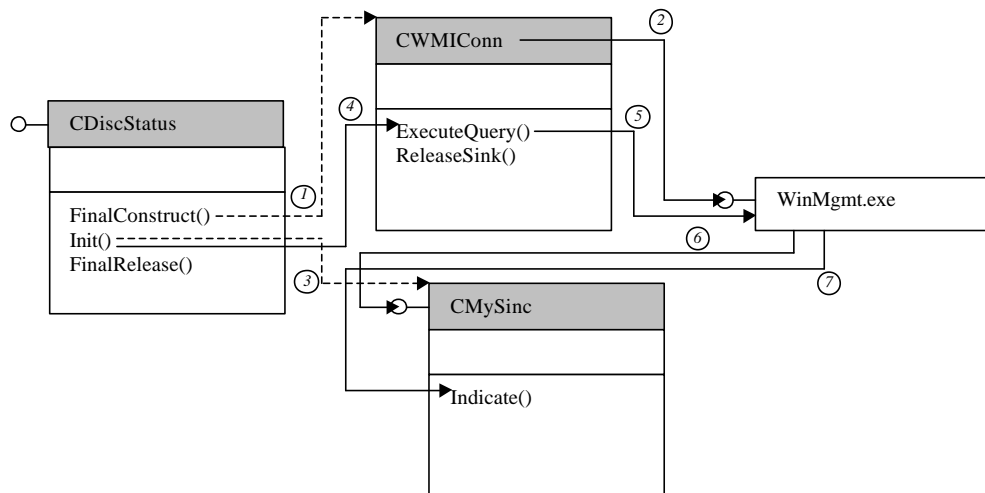


Figure 6-3. Calling schema for the connection to WMI Server.
For efficiency not all methods are shown and no member variables at all are shown.

- 1) After `CDiskStatus` is instantiated its function `FinalConstruct()` is called by its creator. In this method `CDiskStatus` instantiates a `CWMIConn` class.
- 2) The `CWMIConn` constructor connects to the service `WinMgmt.exe` via DCOM as described above in section 6.3.2.

-
- 3) In the `Init()` function of the `CDiskStatus` class one event-sink is then instantiated per query and there are one or more queries per queried WMI Property. If there are no queries available the call chain is broken here.
 - 4) After the sink is created it is passed along with the actual query to the method `ExecuteQuery()` in the `CWMIConn` instance.
 - 5) `CWMIConn::ExecuteQuery()` just passes the parameters on to the service `WinMgmt.exe`.
 - 6) `WinMgmt.exe` then connects to the instance of `CMySinc` that is passed to it in step 5.
 - 7) `WinMgmt.exe` can now send event notifications to the Sink by calling the function `CMySinc::Indicate()`.

7. Implementation of a Simple WMI Provider

This chapter covers WMI Provider writing in general and specially the implementation details of a sample provider, the ABBGenerator.

7.1 Provider

As can be seen in Figure 2-2 and Figure 4-2, the provider is a software layer between the managed element and the WinMgmt service. The provider exists to let WinMgmt access all devices in a uniform way. The provider is attached as an in-process object to the WinMgmt service. When a management application instantiate a class the corresponding provider DLL is loaded and attached, WinMgmt then call `CProvider::EnumerateInstances`. The `EnumerateInstances` method then access the device driver or some other information source and returns the instances that exist. The properties of the instances returned are updated according to the current state of the managed element.

The simple WMI Provider developed here does not access any outer source; it just has an internal representation of three signal generators. The functionality and implementation of the signal generator provider will be covered in greater detail in section 7.3.

The provider developer does decide what functionality to implement, but some methods are necessary for the provider to function at all, like for example `EnumerateInstances`, and `GetObject`. Other methods are used if implemented and the functionality is otherwise unobtainable like `PutInstance`, and `ExecMethod`. Other methods still are, if not implemented emulated by WinMgmt using the methods implemented. The third type of methods is implemented only for performance purposes. An example of such a method is `ExecQuery`, used to get the properties of an instance by querying for it. This can be a performance booster if for example an instance or a property is rather expensive to collect, an example of this are when enumerating the class `Win32_LogicalDisk`, among this class instances are the removable media disk drive, and as every that has used it knows it is rather slow. Excluding this instance when enumerating the class will enhance the performance. If on the other hand if `ExecQuery` is not implemented, WinMgmt will use `EnumerateInstances` anyway and the performance increase by using querying in the management application will be zero.

7.2 MSDN Tutorial

Microsoft Developer Network contains a lot of information on how to design and implement WMI providers with different functionality. A good starting point is the step-by-step guide to developing a provider with basic functionality. The provider tutorial can be found in MSDN Library – April 2000 under the headline “Framework Tutorial”. This provider provides a set of software reindeers, i.e. the reindeers does not map to any

real-world entities. The tutorial was used to develop the ABBGenerator Provider described below, in section 7.3.

7.3 ABBGenerator Provider

The ABBGenerator provider was developed to more thoroughly test the functionality implemented in the Operate^{IT} WMI Aspect. A second objective was to investigate if developing a provider was difficult, and required a lot of special purpose knowledge.

7.3.1 Functional Description

The ABBGenerator Provider should be able to generate three types of signals, sine wave, rectangular wave, and constant increasing signal. The update rate should be constant and independent of the surrounding environment, especially uncorrelated to the management application used to access the value of the signal. The user should be able to set the frequency, of the sine and rectangular wave, from a WMI enabled management tool that includes functionality to set properties. Other properties that the user should be able to modify in the same manner is the amplitude, signal noise, and the status of the generator. The phase shift of the sine wave is another property that the user should be able to modify.

7.3.2 Implementation

The development was a pretty straightforward action following the steps in the tutorial mentioned above, where mainly the class names and the properties were changed. The file that describes the WMI Provider called the Managed Object File, MOF, for the AbbGenerator provider can be seen in Listing B-1 in appendix B. The most evident difficulty was to have a constant tick rate that was independent of the calls made to the provider. If the rate that would be used to pull the values would be known, this could be used but that would violate the requirements above. The two most evident solutions were to, (1) setup a timer that would tick with a known frequency, and (2) spawn a new thread that would tick at a constant rate. The result would be the more or less the same, and the choice was in favor of the second. The thread was spawned in the object constructor, which is run once at provider load time.

```
CAbbGenerator::CAbbGenerator (LPCWSTR lpwszName, LPCWSTR lpwszNameSpace ) :
    Provider(lpwszName, lpwszNameSpace)
{
    DWORD dwTID;

    m_dwTick = 0;
    m_fUpAndRunning = true;

    m_hThread = CreateThread(NULL, 0, TickerFactory, this, NULL, &dwTID);

    SetThreadPriority(m_hThread, THREAD_PRIORITY_TIME_CRITICAL);
    return;
}
```

Listing 7-1. *CAbbGenerator constructor in C++.*

As can be seen in the constructor above, Listing 7-1, the thread is created with the same stack size as the parent thread, and the parent object is passed to the thread in the fourth parameter. After the creation of the thread, the parent sets the priority for the recently created thread to `THREAD_PRIORITY_TIME_CRITICAL`. In Windows 2000 this means that the priority is set to 15 if the process the thread is created in are not running in `REALTIME_PRIORITY_CLASS` in witch case the priority are set to 31. This is done to make the time between two ticks more constant.

```
DWORD WINAPI TickerFactory(LPVOID pArg)
{
    ((CAbbGenerator *)pArg)->SetThreadAlive();

    while (((CAbbGenerator *)pArg)->AmIUpAndRunning())
    {
        Sleep(500);           // Provide a tick every half second
        ((CAbbGenerator *)pArg)->IncTick();
    }

    ((CAbbGenerator *)pArg)->SetThreadDead();

    return WBEM_NO_ERROR;
}
```

Listing 7-2. *The thread starts in the global function TickerFactory.*

As can be seen in Listing 7-2, the code above, the thread announces to the parent object that it is alive when it starts, and that the thread is not alive before it is about to terminate itself. At every loop the thread checks if it should terminate itself, this condition will be false when the parent object's destructor is run. The value of the signal generator is only calculated when asked for by the management application.



8. Evaluation of the WMI Client-Provider Solution

This chapter describes the functionality of the WMI Client-Provider implemented in chapter 6 and 7 of this document. This description will be more focused on the functionality of the WMI Provider and the WMI Client from a users point of view, where as chapter 6 and 7 focuses more on how they are implemented. How the pieces of this chapter fits together can be seen in Figure 2-2, but the box at the bottom “Managed System” does not exist.

8.1 WMI Provider

The WMI Provider is not a visible product. It is linked to the WinMgmt.exe process when needed and does not contain any GUI elements. This gives that it cannot be perceived by the user directly only through management applications. One of the few ways to get direct access to the WMI Provider is to attach a debugger to the WinMgmt.exe process when the provider is linked. A more visually appealing way is still to use a management application. In Figure 8-1 a management application, WMI CIM Studio can be seen. WMI CIM Studio is delivered with the Microsoft Platform SDK. It runs as an ActiveX inside the Microsoft Internet Explorer. The left panel, i.e. the tree-view, shows the CIM Schema for the namespace, i.e. database, connected to, in this case “root\CIMV2”. In Figure 8-1 the class AbbGenerator is selected and instantiated, one of the instances are selected and displayed in the right panel.

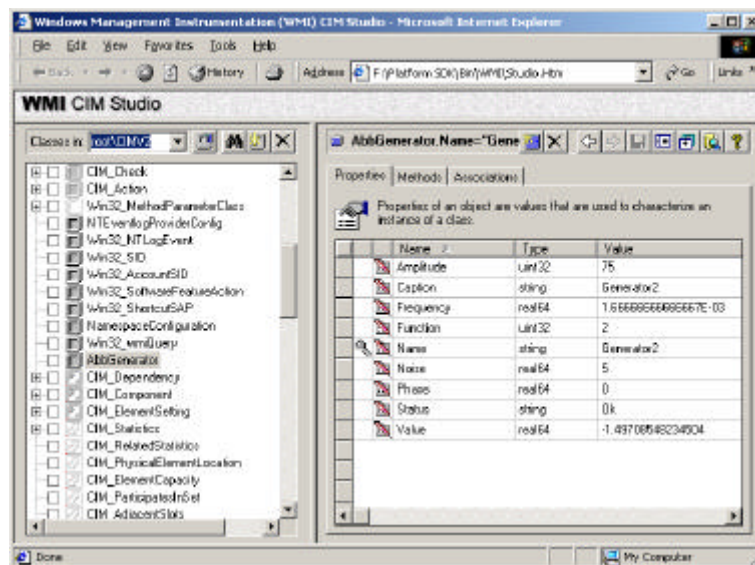


Figure 8-1. WMI CIM Studio, an application developed by Microsoft created to access information handled by the CIM Object Manager.

At the top in the right panel the key property is shown in this case the Name Property, the key property is also denoted by a small picture of a key in the property field. Below the key property is a tabbed view, with the three tabs “Properties”, “Methods”, and

“Associations”. In Figure 8-1 the properties tab is visible, showing the properties of an instantiated AbbGenerator. The properties shown here can be compared with Listing B-1, the MOF file for the AbbGenerator, as can be expected it is a perfect match.

With WMI CIM Studio it is not only possible to view the values of an instance, it is also possible to change the properties not locked. For example this can be used to set the amplitude and noise level.

8.2 WMI Client

The WMI Client described here is the Operate^{IT} WMI Powered Aspect. This Aspect can opposed to WMI CIM Studio not change the values of WMI Properties. It can rather be thought of as a one-way communication. The WMI Aspect subscribes to changed WMI Properties and makes the value available to the Operate^{IT} Platform as OPC Properties. These subscriptions are setup from the Aspects configuration view, seen in Figure 8-2.

In Figure 8-2 an Aspect Object named MyGen1 is created and the WMI Aspect, named nt_status00, is added. Here two WMI Properties are subscribed to, Status and Value. The Value property is when received directly without any processing published as an OPC Property. But the Status property is when changed processed, if it reports Error then S_STATUS is set to Error and if it reports Degraded S_STATUS is set to Warning this in accordance with the mapping proposed in section 5.3.2, Table 5-1.

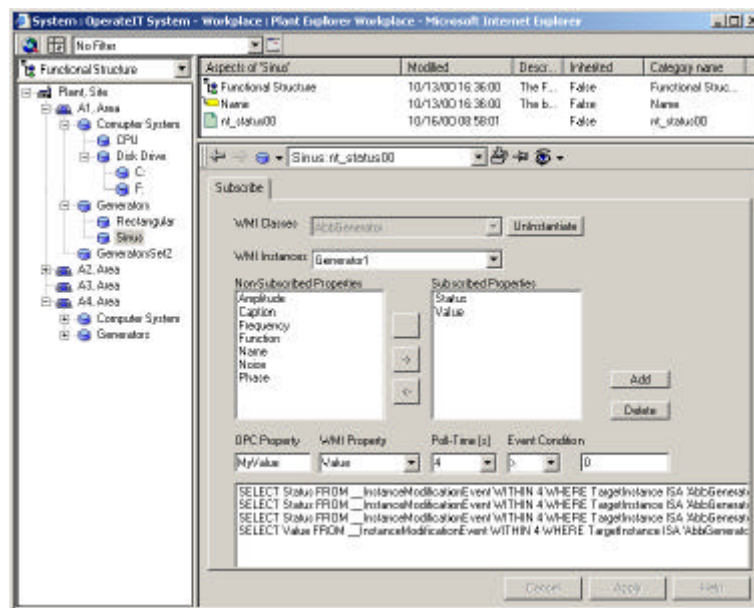


Figure 8-2. Plant Explorer with the WMI Aspects configuration view showing.

The Non-Subscribed Properties plus the Subscribed Properties gives the same properties as can be seen in Figure 8-1 and the MOF file in Appendix B.

8.3 WMI Client-Provider

This section will answer the question of how does the WMI Client-Provider solution function together. The main objective of this project is to supervise the status of standard computer and office equipment. The AbbGenerator has the status property and this can be used to simulate a status change in the managed element. Normally the status property is set by the managed element, this is not so with the AbbGenerator, where the status property is changeable from any management application that allows writing to WMI Properties. The status property is changed to “Error”, “Degraded” and then “Ok” and the result can be seen in Figure 8-3, this after a mapping like the one in Figure 8-2.

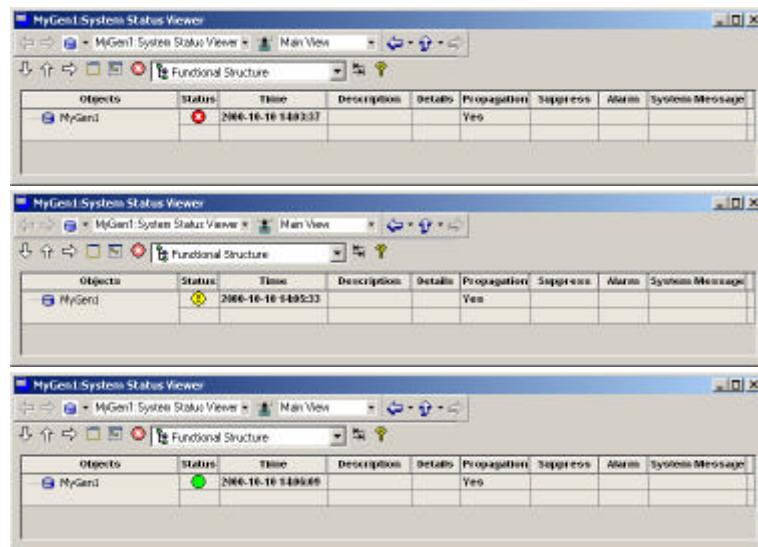


Figure 8-3. The System Status Viewer Aspect showing the status of the MyGen1 Aspect Object, which is powered by the AbbGenerator through WMI. From the top, a) Error, b) Warning, and c) Ok.

The next evaluation test is to publish mapped WMI Properties as OPC Properties. The AbbGenerator WMI Property “Value” is mapped to the OPC Property “MyValue” for two instances of the generator. One instance of the signal generator is generating a sine wave and the other a rectangular waveform. The configuration of the second signal generator can be seen in Figure 8-1, above, as can be seen the amplitude is 75 units, and the frequency $1.667 \cdot 10^{-3} \text{ s}^{-1}$, i.e. period time of 600 s, and a noise level of ± 5 units. The two OPC Properties are plotted in a Trend Display Aspect, as can be seen in Figure 8-4. If one compares the plot with the configuration of the provider the congruence is striking. The average at the top values are around 75 and the period is around 10 minutes or 600 s. The third, not so regular, plot seen in the Trend Display is the current CPU utilization, this information is also received from WMI.

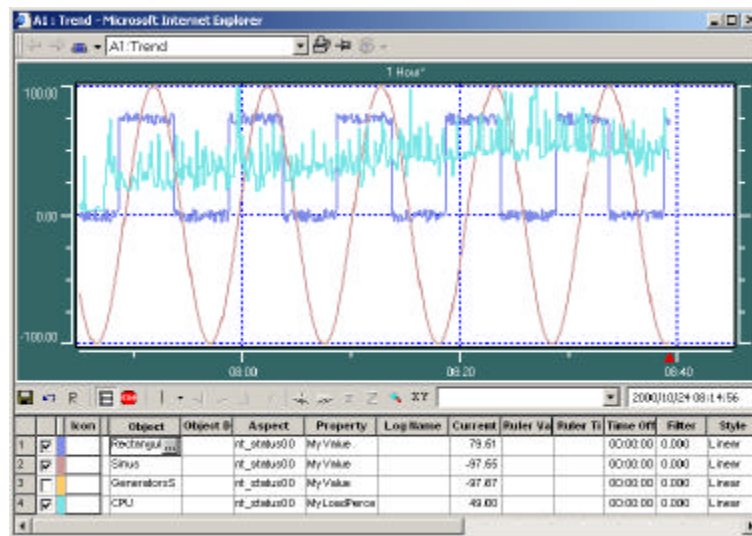


Figure 8-4. A trend display showing OPC Properties published by the WMI Aspect.

8.4 Concluding Remarks

The two main functionalities, supervision of status and generic mapping of WMI Properties functions as expected. The configuration of the Aspect for generic mapping of WMI Properties to OPC Properties functions rather well even though not all functionality seen in the GUI are present, like removal and editing of queries. Also the WMI Provider implemented in this project is working as expected.

9. Future Work

This chapter is divided into two parts. Part one covers the evolution of the WMI Powered Aspect, and part two covers how WMI might be used in future products.

9.1 Improved WMI Aspect

The basic functionality of the WMI Powered Aspect prototype worked well, if used in the correct way. But the prototype lacked some necessary features. Example of such features is the possibility to edit and delete queries used for retrieving WMI Properties. Another necessary improvement is to streamline the user interface. The prototype actually contained a number of graphical elements that could be removed, and still other graphical elements could be replaced for a better and more easily comprehended user interface. An example of an improved user interface will be detailed below.

For maximum rapidity of the development of the prototype the data handling was at several places implemented where the data was most easily accessed. This in opposition to Model-View-Controller design, where separation of the model, where the data processing is implemented, and the view where the data is presented, is done. For example the Add button in the configuration view implements the complete functionality to build a query using WQL. In a more thought-trough design the functionality to build the query should reside in a dedicated function, this function would build the query from parameters passed to it.

In Figure 9-1 a layout of an improved user interface for the configuration view can be seen. This layout should be compared to the layout in Figure 5-5, the pre-study layout, and the one in Figure 8-2, the actual implemented layout.

Active	OPC Prop.	WMI Prop.	Type	Override Name	Poll-rate	Condition
<input checked="" type="checkbox"/>	CPU Load	LoadPercent	uint16	<input type="checkbox"/>	4	> 10
<input checked="" type="checkbox"/>	Name	Name	string	<input checked="" type="checkbox"/>	30	none
<input type="checkbox"/>				<input type="checkbox"/>		
<input type="checkbox"/>				<input type="checkbox"/>		

Figure 9-1. What an improved user interface could look like.

The most interesting comparison is between the new layout and the implemented layout, this since the implemented is an improvement of the pre-study layout. The comparison gives that all buttons except the “Instantiate”/“Uninstantiate” are removed, this to not confuse the user and to avoid unnecessary and time-consuming clicking. Noteworthy is that the buttons “Cancel”, “Apply”, and “Help” are inherited from the standard functionality of the aspect configuration view and not subject to review here.

Also the two list-boxes in the middle of the view are removed, this because their function only was to give an overview of the WMI Properties possible to subscribe to. Instead all the WMI Properties are shown in the field called "WMI Prop." in Figure 9-1 when it is selected. This is possible if this field is a "hidden" combo-box, i.e. a combo-box when selected and a text-field when not selected. The "hidden" combo-box would when selected show all WMI Properties to let the user select what property to subscribe to. When not selected the subscribed property would be shown. The major change is that the multiple fields at the bottom of the configuration view in the implementation have been replaced with a single list-view with multiple fields. The new list-view is taking care of all the functionality of the old elements and a little additional functionality is also added, like configuration of the overriding of the name property of the Aspect Objects name aspect.

The first field in the list-view is a selection-box telling if the subscription to the WMI Property is active. The second field is the published name of the OPC Property. The third field is as described above a combo-box showing all the WMI Properties and a feature that should be implemented is that when a valid selection is done, default values for the other field should be presented. This feature makes the standard configuration faster and less error prone. It should however not overwrite and already entered data. The next field is a read-only text field telling the type of the WMI Property selected. The fifth field is a radio button that is only active if the property is of string type. This field is optional but should if implemented override the name property of the Aspect Objects name aspect. The remaining fields are in essence the same as the ones in the implementation just put in a list-view.

Removing an entry is done just by selecting the entire entry and clicking the delete key on the keyboard. Editing is done by selecting the field and making the change. All changes, i.e. Adding, Deleting, and Editing, are confirmed and performed by clicking the Apply button.

Ideally five clicks with the mouse should be sufficient to setup a subscription to a WMI Property. After the initial setup, adding more subscriptions requires a minimum of only two clicks. These numbers builds on the assumption that one is satisfied with the default values presented in the other fields.

9.2 WMI Related Extensions

This project evolves around the objective to supervise and manage standard computer and office equipment. This is a natural way of using WMI but other applications could also be thought of, like for example to present Operate^{IT} specific data, like user groups, using the CIM. Thus making it available to all CIM enabled management applications. One immediate benefit of this would be that building configuration snap-ins for MMC would be easy.

It might also be possible to map the entire ABB Aspect Object Model using the CIM. But this would require extensive research on how to do it, what benefits would be achieved, and if it at all would be desired.

10. Conclusion

10.1 Fulfillment of the Objectives

In this section will validate the fulfillment of the objectives and where in the document the particular objective is fulfilled.

Objective 1

This project opts to make it easy to supervise and manage standard office and computer equipment using the existing ABB Operator Station, Operate^{IT}.

The main sources of information are chapter 6, and chapter 8. But the other chapters work to build a foundation for the fulfillment of this objective.

Objective 2

Examine WBEM and describe Microsoft's implementation, WMI.

The first part of the objective is covered in section 4.1, especially 4.1.2, and the second objective in section 4.2.

Objective 3

Investigate how the CIM-schema can be extended with ABB Objects.

This objective is not fulfilled but why is described in section 9.2.

Objective 4

Investigate if it is possible, and what is required, to make Operate^{IT} specific data available to third-party software via WMI with the help of a WMI Provider.

The implementation of a WMI Provider is done detailed in chapter 7, especially in section 7.1 and 7.3. Also chapter 8 contains additional information.

Objective 5

Examine the support WMI is getting from third-party developers.

The support from third-party developers are investigated briefly in section 4.3.

Objective 6

Investigate how to map the CIM into the ABB Aspect Object Model.

This is covered in great detail in chapter 5, especially section 5.3.

10.2 Concluding Remarks

The Operate^{IT} Platform is a newly designed and implemented supervision and management platform for factory automation. This platform has many advantages over its predecessors and competitors. The advantages include, 1) it is fundamentally based on an object-oriented model that is powerful and easy to use and comprehend, 2) it incorporates structures for the possibility to easily navigate through information, 3) it is built on standard components, and 4) it is extendable with standard components.

As more standard computer and office equipment are brought into the factories, for a reliable production, supervision of these components are inevitable. The only viable solution to supervise the innumerable large amount of different components on the market today is to use the WBEM standard brought forward by DMTF. The Operate^{IT} Platform is currently only run on Windows 2000. Luckily Windows 2000 has support for WBEM through Microsoft's implementation WMI. The WBEM standard is designed so that it incorporates a number of older standards and therefore the support for WBEM are in some areas already well developed. But also since WMI builds upon providers developed by the constructor of the managed element not all managed elements has a provider and some of the providers that exists are lacking in functionality. Still it is more than likely that the support for WBEM will grow rapidly.

WBEM standard and the Microsoft implementation WMI are easy to comprehend and use. Although some issues existed with WMI, this because the connection to it is done through DCOM. One of the ideas behind the COM/DCOM concept is that one should not have to care about where the called function is executing, in-process or out-of-process. This is not entirely true, if for example the string handling is done incorrect the error will be hard to track and also the security has to be setup correct or strange run-time errors will be obtained.

Appendix A – Description of Abbreviations and Words

.NET	Concept designed by Microsoft to make the Internet a distributed computing platform
ACP	ABB (or, Advant, Automation) Control Platform, the word is obsolete
ADO	ActiveX Data Object
Afw	Aspect Framework
AOM	ABB Object Manager
ASO	Aspect System Object
ATL	Active Template Library
CIM	Common Information Model, standard by DMTF
CIMOM	CIM Object Manager
COM	Component Object Model
COTS components	Common of the shelf components. Components not specifically designed for the industry, but rather for the desktop and server market.
DCOM	Distributed COM
DEN	Directory Enabled Network
DLL	Dynamic Link Library, library loaded at run-time when the code or data contained therein is needed
DMI	Desktop Management Interface, DMTF standard
GUI	Graphical User Interface
GUID	Globally Unique Identifier, 128 bit number unique in both time and space
HTA	HTML Application
IDL	Interface Definition Language, standard by OSF
IID	Interface Identifiers
Interface	In this document COM Interface, if not found right behind 'user'
LDAP	Light-weight Directory Protocol
MMC	Microsoft Management Consol, delivered with Windows 2000, supposed to handle all configuration of the computer system including software
MOF	Management Object Format
MSMQ	Microsoft Message Queue
NLS	National Languge Support
OPC	OLE for Process Control
OSF	Open Software Foundation
RPC	Remote Procedure Call, standard defined by OSF defining how a procedure on a remote system should be accessed and how to return data.
SNMP	Simple Network Manage Protocol
SQL	Structured Query Language
UML	Unified Modeling Language, is a graphical language for visualizing,

	specifying, constructing, and documenting the artifacts of a software-intensive system. [12]
UPS	Uninterruptible Power Source
USB	Universal Serial Bus
UUID	Universally Unique Identifier, standard by OSF
WBEM	Web-Based Enterprise Management
WDM	Windows Driver Model
Windows DNA	Evolved into .NET
WMI	Windows Management Instrumentation
WQL	WMI Query Language
WSH	Windows Script Host
XML	Extensible Markup Language

Appendix B – MOF file for the WMI Provider AbbGenerator

For the WinMgmt.exe to know what providers are available and what to expect from them a MOF file is used to describe the WMI Provider. Below in Listing B-1 the MOF file for the AbbGenerator can be seen. To register the provider, use the MOF compiler, MOFComp.exe, provided with the Microsoft Platform SDK. The compiler will parse the file and check for errors and then store the data in the CIM Repository.

```
#pragma namespace ("\\\\.\\ROOT\\CIMV2")

instance of __Win32Provider as $P
{
    Name = "AbbGenerator";
    ClsId = "{2303e004-98e4-4fe0-a584-3b14608a0d72}";
};

instance of __InstanceProviderRegistration
{
    Provider = $P;
    SupportsGet = TRUE;
    SupportsPut = TRUE;
    SupportsDelete = FALSE;
    SupportsEnumeration = TRUE;
    QuerySupportLevels = {"WQL:UnarySelect"};
};

instance of __MethodProviderRegistration
{
    Provider = $P;
};

[Dynamic, Provider ("AbbGenerator")]
class AbbGenerator
{
    [read, key]    String Name;
    [read]        String Caption;
    [read]        Real64 Value;
    [read, write] uint32 Amplitude;
    [read, write] uint32 Function;
    [read, write] Real64 Frequency;
    [read, write] Real64 Phase;
    [read, write] Real64 Noise;
    [read, write] String Status;
};
```

Listing B-1. The MOF file that is registered with the CIM Object Manager.

Appendix C – WMI Aspect User Guide

Even though the final product developed by this project is a prototype it might be used for demonstration purposes and similar types of events. This user guide will, first try to describe the user interface and how to use it, then be a guide, on a step-by-step basis, on how to expose a single OPC Property, and at the end describe the most evident pitfalls and how to avoid them.

C.1 User Interface Components

The user interface is built on standard graphical elements found in the Microsoft Windows environment. The elements used in this configuration view are listed in Table C-1, along with what they look like and how they are used.

Graphical Element	Appearance	Usage
Label	Text on a flat gray background	Show static information
Button	3D Envelope with text	Receive user actions
List-Box	3D Cutout and white area with a number of single lines of text	Multiple selection with predefined choices
Text field	3D Cutout and white area with continuous text	Custom data input
Combo-Box	Text field + button with an arrow, when the button is pressed a list-box is shown below	Single selection with predefined choices, space economical solution

Table C-1. Some graphical elements and how they are used in this project.

The graphical elements described above are assembled into a user interface for the configuration view. This configuration view can be seen in Figure C-1. In this figure numbers from 1 to 16 are added to all the graphical elements except the label elements. This is to make it possible to reference them in following text sections. Labels are not numbered; this because these elements are not usually interactive, they are just information carriers. Also the “Cancel”, “Apply”, and “Help” buttons are not numbered, this because they are not unique to the WMI Aspect, they exist and functions the same in all aspect views. The numbering is done from top to bottom and left to right as suitable.

Below are all the interactive elements described. The description consists of, how they are used, where they get their information from and where they output any data if they do.

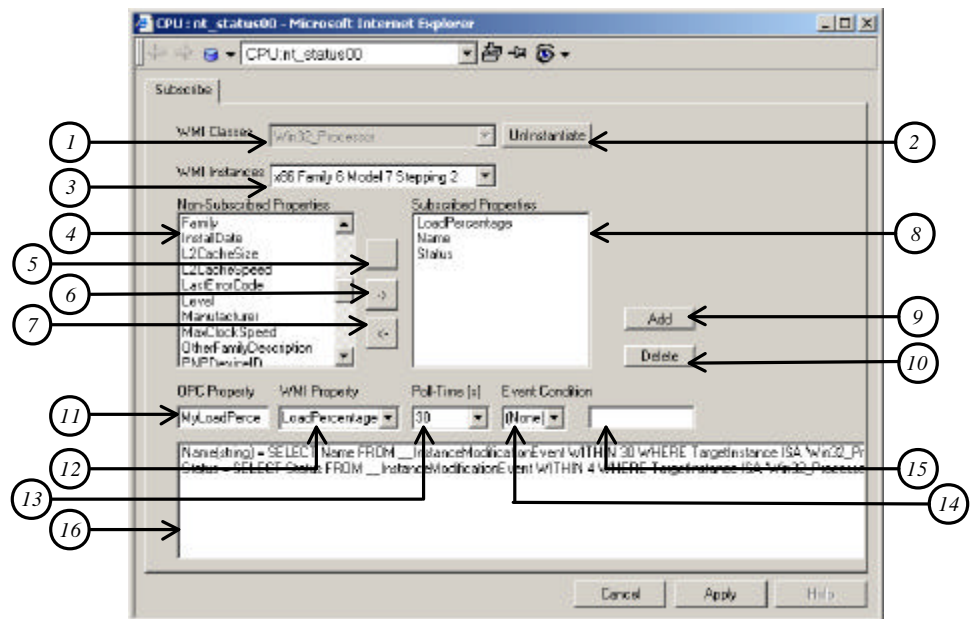


Figure C-1. The configuration view of the WMI Powered Aspect.

- 1) Decides what WMI Class this Aspect Object should correlate to. The name of the Aspect Object should somewhat give a hint to what WMI Class is instantiated here. The classes supported in this prototype can be seen in Table C-2.

AbbGenerator

Win32_CDROMDrive

Win32_ComputerSystem

Win32_DiskDrive

Win32_LogicalDisk

Win32_OperatingSystem

Win32_PerfRawData_PerfProc_Process

Win32_Printer

Win32_Process

Win32_Processor

Table C-2. *A selection of the WMI Classes.*

Most of the classes are self-explanatory, but two classes will be described in more detail the class `AbbGenerator`, and `Win32_PerfRawData_PerfProc_Process`. `AbbGenerator` is a class created in the same project as the WMI Aspect to test the functionality of the Aspect. `Win32_PerfRawData_PerfProc_Process` is a class receiving data from the low-level performance counters built into the Windows NT/2000 kernel.

- 2) A button used to lock and unlock the currently selected WMI Class. Here, since the class `Win32_Processor` is instantiated, the caption of the button says "UnInstantiate" but if no class was it would say "Instantiate".
- 3) In this combo-box the names of the instances that exist in the namespace are entered. And the first in the list are selected by default.
- 4) List-box showing all the not yet subscribed WMI Properties. Naturally this list-box shows all properties when the class is first instantiated.
- 5) "Wizard Button", this button does the same as (6) but also adds default values to the fields (11), (12), (13), (14), and (15).
- 6) Button used to subscribe to a WMI Property. When this button is clicked the property marked in (4) are moved to (8) and (12).
- 7) Button used to unsubscribe a WMI Property. When this button is clicked the property marked in (8) are moved to (4) and removed from (12).
- 8) List-box showing what properties are subscribed to. Always in synchronization with (12).
- 9) Used to add queries, i.e. the fields (11), (12), (13), (14), and (15) are filled with valid information. The query is then showed in (16) for debugging purposes.
- 10) Obsolete. Intended used, to remove queries. Not implemented in this prototype.
- 11) Text field where the name used for the publication of the OPC Property are entered. Default (5): "My" + <WMI Property Name>.
- 12) List-box showing what properties are subscribed to. Always in synchronization with (8). Default (5): Showing the latest added property.
- 13) Combo-box with the poll-rate used to poll the WMI Property in seconds. Values to select from are {1, 2, 3, 4, 5, 10, 15, 30, 60, 120, 500}. Default (5): 30.
- 14) Combo-box with the event condition operator used to poll the WMI Property. This together with (15) makes up the event condition. Values to select from are {(none), =, <, >, <=, >=, !=}. Default (5): (none).
- 15) Text field with the event condition value. This together with (14) makes up the

event condition. Only used if (15) is different from (none). Default (5): blank.

- 16) Shows all queries used. Queries are added to this list-box when button (9) is clicked. This is so that the user can see what queries are active on this aspect, and for debugging purposes.

C.2 Tutorial

This section covers the configuration of a WMI Aspect in a step-by-step fashion. The requirements put upon the user are virtually none other than some normal Operate^{IT} know-how but on the other hand is the learning value rather low. The tutorial is designed so that anyone who fulfills the requirement should be able to follow it.

The tutorial describes how to add the supervision of the CPUs utilization as an OPC Property. This is performed on a single CPU system with a Pentium III processor. In this scenario the user is only interested of the CPU utilization when it is above 50%. All graphical elements will be referenced by the number given to them in Figure C-1.

Step 1

Q: What managed element do we want to supervise?

A: The CPU.

Q: What is the CIM Class covering the supervision of the CPU?

A: The Win32_Processor.

TODO: Select Win32_Processor in (1) and click (2).

Step 2

Q: Which CPU does we want to supervise?

A: Only one is available, "x86 Family 6 Model 7 Stepping 2".

TODO: Nothing, the correct instance are already selected.

Step 3

Q: What property tells us of the CPU utilization?

A: The property "LoadPercentage".

TODO: Add it to (8) by clicking the 'wizard' button (5). This will take us to the situation shown in Figure C-1, just that in that figure the properties Name and Status is also supervised.

Step 4

Q: Is not 30 s between each poll a bit slow?

A: Might be, lower it to 4 s.

TODO: Set (13) to "4", (14) to ">", and enter "50" in (15). Figure C-2 shows the result.

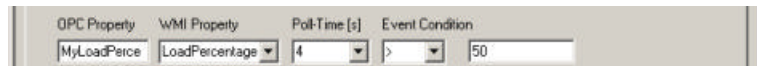


Figure C-2. The query build bar, after the default values are customized.

Step 5

TODO: Click the add button (9) and then the apply button. The query is added.

C.3 Pitfall Avoidance

One of the more problematic things are that ' has to be used around the condition event value when queering for properties of the type string.

References

Due to the lack of maturity in the areas covered in this paper the references table are rather unstructured. The goal has been to follow the structure

AUTHOR (year). Article. *Paper*. Page.

This has been hard especially with the electronically distributed information sources where the paths from the root are in electronic form sometime gibberish, and explicit form extremely long. Where at all possible the electronic form has been chosen for reader convenience.

- [1] Intel Architecture Software Developer's Manual Volume 3: System Programming, ordering number 243192.
- [2] Windows NT Architectural Modules, MSDN as of April.
- [3] HP JetAdmin Reference Manual,
<http://www.hp.com/cposupport/networking/manuals/bpj06492.pdf>
- [4] Web-based Enterprise Management,
<http://www.dmtf.org/download/spec/wbem.ppt>
- [5] WESTERINEN, A., STRASSNER, J. (2000). Common Information Model (CIM) Core Model, version 2.4. *DMTF White Paper*, p. 1.
- [6] DTMF (1998). XML As a Representation for Management Information – A White Paper.
- [7] (2000). WAP och Bluetooth – industriell IT med spets. *ABB Kunden*. Nr 2. p. 13.
- [8] ANDERSSON, J. (2000). Goals, Requirements and Background. *Architecture Description Industrial^{IT}: Workplace and Server Architecture*, p. 10.
- [9] SOMMERVILLE, I. (1995). Software Engineering. 9.
- [10] WESTERINEN, A., STRASSNER, J. (2000). Common Information Model (CIM) Core Model, version 2.4. *DMTF White Paper*, p. 20.
- [11] SVENSSON, K. (2000). Description of Function ABB Automation System Application C++ Wizzard. Revision 1.
- [12] BOOCH, G. (1998). The Visual Modeling of Software Architecture for the Enterprise. MSDN Library October 2000.
- [13] SVENSSON, A. (2000). Description of Function System Status. 3BSE018730. Rev. B.
- [14] SVENSSON, A. (1999). 2.4 Basic Status Properties. *Programmer's Guide System Status*. Page 5.

