INSTITUT FÜR INFORMATIK
DER LUDWIG-MAXIMILANS-UNIVERSITÄT

INSTITUT FÜR INFORMATIK
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Prof. Dr. Heinz-Gerd Hegering

Prototypischer Einsatz von WBEM für das Management von Windows NT

Bearbeitende:
Bernd Süß (suess@informatik.tu-muenchen.de)
Martin Kacalek (kacalek@informatik.tu-muenchen.de)

Betreuer:
Rainer Hauck

Inhaltsverzeichnis

1 Prolog

Vor 2 Jahren wurde von den Firmen BMC Software, Cisco Systems, Compaq Computer, Intel und Microsoft die Web-Based Enterprise Management (WBEM) Initiative ins Leben gerufen. Ziel dieser Initiative war es einen einheitlichen Mechanismus zur Beschreibung und zum Austausch von Managementinformation zu entwickeln. Heute wird diese Initiative von mehr als 70 namhaften Herstellern wie HP, Computer Associates und IBM/Tivoli unterstützt.
Derzeit gibt es für Administratoren heterogener Umgebungen keine einfache, integrierte Möglichkeit, Informationen über den Zustand der verschiedenen verwalteten Geräte („Managed Object“, MO) zu erlangen. Meist ist für jedes dieser MOs ein eigenes Programm oder Administrationswerkzeug nötig.
WBEM macht es möglich eine einheitliche Oberfläche für die verschiedenen Plattformen, Komponenten und Applikationen im zu verwaltenden Netzwerk zur Verfügung zu stellen.
Ziel dieser Arbeit ist es, eine Evaluierung des von Microsoft entwickelten WBEM Software Development Kits (SDKs) anhand einer prototypischen Modellierung von Managementinformationen durchzuführen.
Zur Verfügung stand dabei nur eine Betaversion der Software. Während der Evaluierungsphase wurde eine neue Version des SDKs, leider immer noch eine Beta Version, veröffentlicht. Zeitgleich dazu ist die Entwicklungsarbeit an den Konzepten zu WBEM an die Desktop Management Task Force (DMTF) übergeben worden. In der Folge dessen wurde schnell klar, daß viele der in der ersten Beta Version enthaltenen Konzepte noch einmal völlig überarbeitet und zum Teil völlig neue Konzepte eingeführt wurden. So wird die Endgültige Version des SDKs keine Java APIs mehr enthalten und sich nach Aussagen von Microsoft statt dessen auf XML über HTTP abstützen.
Ein Hauptproblem bei der Arbeit mit dem WBEM SDK war die mangelhafte und zum Teil nicht vorhandene Dokumentation. Durch den Versionswechsel mitten in der Evaluierungsphase wurde das Testen zusätzlich erschwert. Funktionierende Teilbereiche waren in der neuen Version nicht mehr lauffähig, andere Teilbereich mußten mehr oder weniger neu evaluiert werden. Auch die vorhandenen Newsgroups und Mailinglisten haben meist keinerlei Antwort hervorgebracht.

2 Historie

Die Entwicklung des WBEM Standards begann 1996 unter der Initiative von namhaften Firmen wie BMC Software, Cisco, Compaq Computer, Intel und Microsoft. Während Microsoft mit der Entwicklung eines SDKs für WBEM Applikationen und Provider begann, versuchte das Konsortium als Untergruppe der Desktop Management Task Force (DMTF) die Entwicklung eines Informationsmodells für WBEM voranzutreiben. Die Entwicklung eines Standards in den Bereichen Organistations-, Kommunikations- und Funktionsmodell wurde fast völlig von der Firma Microsoft übernommen. Es ist daher nicht verwunderlich, daß vor allem die wirtschaftlichen und technologischen Interessen dieser Firma auf die Entwicklung Einfluß genommen haben. So fällt es heute auch schwer zwischen einem WBEM Standard und der Implementierung dieses Standards von Microsoft zu unterscheiden, zumal sämtliche Diskussions- und Konzeptpapiere zum WBEM Standard von Mitarbeitern der Firma Microsoft stammen. In dieser Arbeit soll vor allem auf die Fähigkeiten des MS WBEM SDKs eingegangen werden. Der WBEM Standard wird nur in den Kapiteln 3 und 4 berücksichtigt.

Mitte 1997 wurde von Microsoft eine erste Version des Microsoft WBEM Software Development Kits (MS WBEM SDK) als Beta1 veröffentlicht. Die DMTF hatte kurz vorher das CIM (Common Information Modell) Informationsmodell in der Version 1 standardisiert. Dieses MS WBEM SDK basierte auf dem im CIM-Standard verabschiedeten Informationsmodell.

Im Oktober 1997 wurde dann nach einer Weiterentwicklung des MS WBEM SDKs eine weitere Betaversion veröffentlicht. Diese Version brachte verschiedene Erweiterungen, außerdem wurden einige in der Beta1 vorhandene Fehler ausgebessert.
Mit der Verabschiedung des CIM Informationsmodells in der Version 2.0 wurden auch mehrere Änderungen in der Implementierung des SDKs nötig. Ursprünglich war die Version 1.0 des SDKs für Anfang 1998 angekündigt, durch die nötig gewordenen Änderungen ist dieser Termin auf Mitte des Jahres 1998 verschoben worden.

Am 2. Juni 1998 wurde die gesamte Initiative unter die Aufsicht der DMTF gestellt. Zeitgleich wurde auch ein neuer Build des WBEM SDKs von Microsoft veröffentlicht. Dieser Build basiert auf der Version 2 des CIM Informationsmodells und integriert unter anderem zum ersten Mal einen DMI Provider.

Gegen Mitte Juni 1998 wurde das WBEM SDK nach Aussagen von Microsoft in die Produktion übergeben. Die Auslieferung ist über das Developer Network Programm (MSDN) der Firma Microsoft mit dem Juli 1998 Update geplant.

3 Informationsmodell

Die Gründungsmitglieder der WBEM Initiative hatten zwei Hauptziele. Das erste war eine standardisierte Möglichkeit, Managementinformationen zu sammeln und einer Managementapplikation zur Verfügung zu stellen (mehr Informationen dazu unten).
Das zweite Ziel war eine Standardisierung der Darstellung dieser Informationen.
Um dies zu erreichen wurde von der Desktop Management Task Force (DMTF) 1997 das Common Information Model (CIM) entwickelt. CIM ist ein objektorientiertes Schema zur Beschreibung der Management Objekte eines Systems. Mit CIM können Daten von Netzkomponenten, Arbeitsstationen oder einzelnen Hardwarekomponenten, aber auch Applikationen, Peripheriegeräte oder z.B. Datenbanken beschrieben werden.
CIM unterstützt die in der Objektorientierung üblichen Mechanismen wie Vererbung und Methoden. CIM ist unabhängig von jeder Programmiersprache und der Umgebung in der es genutzt wird.
Zu diesem Zweck wurde ein Meta Schema entwickelt. Dieses Schema legt genau fest, wie Informationen in CIM strukturiert werden. Darauf aufbauend wird auch ein Dateiformat definiert. Das Managed Object Format (MOF) stellt die Information über die MOs, die mit CIM modelliert werden in maschinenlesbarer Form da.[1]

3.1 Aufbau des Common Information Model

Das Common Information Model besteht nicht aus einem einzigen Schema, sondern aus mehreren auf dem Meta Schema basierenden Schemata. Der Zusammenhang zwischen diesen Schemata wir durch Abbildung 3.1 verdeutlicht.

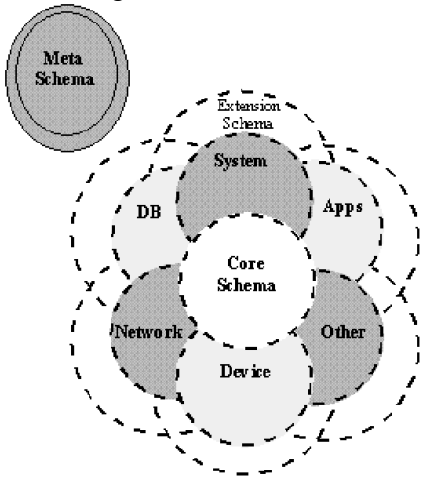

Abbildung 3-1: CIM Schemata

Im Meta Schema werden die grundlegenden Regeln für die dem CIM zugrunde liegenden Modellierungssprache festgelegt. Im Core Schema werden Informationen, die auf alle Aspekte des Managements anwendbar sind, modelliert. Die Common Schemata, wie das Device oder das Network Schema, erweitern das Core Schema um

[1] Nach [Todd]

domänenspezifische Informationen. Die Meta-, Core- und Common-Schemata sind von der DMTF standardisiert.
Die verschiedenen Common Schemata können durch sog. Extension Schemata speziell z.B. nach Herstellervorstellung erweitert werden.
Die Komplexität des gesamten Modells würde den Rahmen dieser Ausarbeitung sprengen und kann in der von der DMTF veröffentlichten Literatur nachgelesen werden. Hier wird zum Verständnis des Aufbaus der von uns modellierten Information kurz auf das Meta Schema eingegangen.[2]

3.1.1 Meta Schema

Das Meta Schema liefert die Bausteine zur formalen Beschreibung der Managementinformation und beschreibt die Semantik des Modells. Die Technik der Objektorientierung findet im gesamten Modell Verwendung. Aus Abbildung 3-2 läßt sich ein Überblick über das Schema gewinnen.[3]

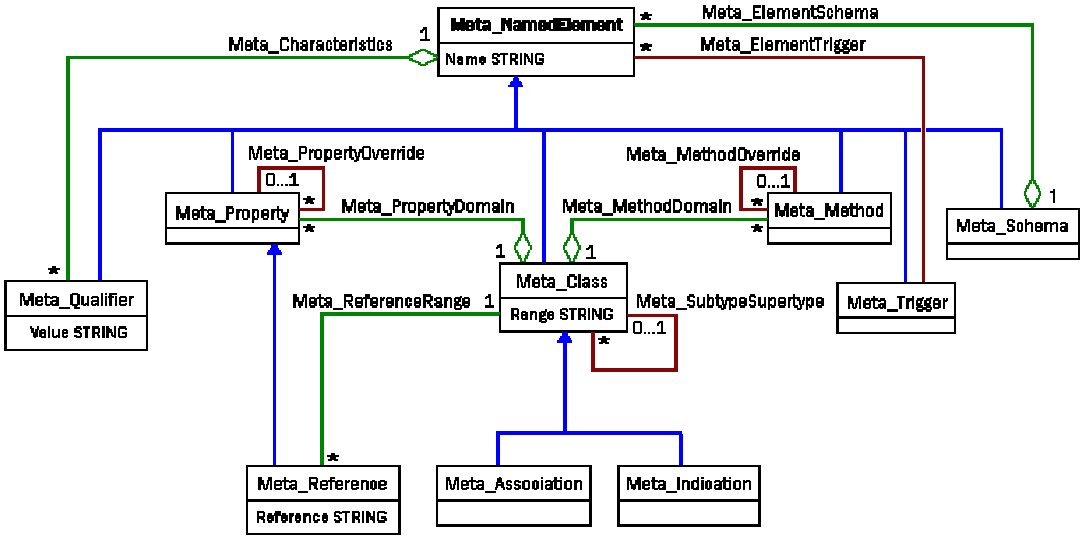

Abbildung 3-2: CIM Meta Schema

Folgende zentralen Objekte werden im Meta Schema definiert:

Klassen

Klassen sind der zentrale Bestandteil der CIM Schemata. Sie sind eine Beschreibung der MOs durch die verschiedenen Attribute und Methoden dieser. Gleichartige MOs werden als Instanziierungen derselben Klasse modelliert. Beispielsweise können alle in einem Netzwerk vorhandenen Router als Instanzen einer Routerklasse dargestellt werden. Durch Verwendung der Objektorientierung können Vererbungshierarchien

[2] Nach [CIM1]
[3] Nach [CIM2]

aufgebaut werden. Jede Klasse kann allerdings nur eine Superklasse haben, es ist also keine Mehrfachvererbung möglich.
Klassen können in CIM als „abstract“ definiert werden. Eine abstrakte Klasse ist nicht instanziierbar und dient dem Aufbau von Hierarchien. Konkrete Implementierungen und damit Instanzen dieser Klassen sind erst in einer abgeleiteten Klasse möglich.

Ein Klasse wir durch folgende Komponenten beschreiben:

- Attribute
  unter einem Attribut versteht man einen Werte die eine Klasseninstanz näher beschreibt. Auf diesen Wert kann von außen über PUT und GET Operationen zugegriffen werden. Ein Beispiel für ein Attribut ist der Name einer Klasse
- Methoden
  unter Methoden versteht man eine Signatur (Name, Eingabeparameter, Rückgabewert) d.h. eine Definition einer Funktion auf einer Klasse. Diese Methoden können bei instanziierbaren Klassen eine Implementierung enthalten. Ein Beispiel für eine Methode wäre bei einem Router eine „REBOOT“ Methode die das Gerät bootet.
- Qualifier
  durch Qualifier werden einzelne Objekte genauer charakterisiert. Im Gegensatz zu Attributen sind Qualifier dazu gedacht zusätzliche Informationen, wie z.B. die Art des möglichem Zugriffs oder eine informelle Beschreibung, über diese Objekte zu enthalten. Qualifier können sowohl auf Klassendefinitionen, Klasseninstanzen, Attribute und auch Methoden angewandt werden. Ein Beispiel ist der Qualifier „READ“ der angibt, daß ein Attribut lesbar ist.
  Ein weiteres Beispiel ist der Qualifier „abstract“ der für eine Klasse angibt, daß diese nicht instanziiert werden kann und nur eine abstrakte Definition von Attributen und Methoden für einen speziellen MO-Typ ist. Neben den in den verschiedenen Schemata definierten Qualifiern sind auch benutzerdefinierte Qualifier möglich. Dadurch ist es (wenn auch eingeschränkt) möglich das Metaschema zu erweitern.

Assoziationen

Assoziations Klassen sind eine spezielle Form einfacher Klassen. Sie enthalten mindestens zwei Referenzen. Eine Referenz ist ein Verweis auf eine Instanz einer Klasse. Diese Referenz hat keinen Einfluß auf die referenzierte Klasse.
Eine Assoziations Klasse kann Subklasse einer einfachen Klasse sein. Jede Subklasse einer Assoziations Klasse ist auch wieder eine Assoziation.
Dieser Mechanismus erlaubt es m:n Beziehungen zu modellieren.
Eine Anwendung dieser Technik ist in Kapitel 6 gezeigt.

Namespace

Namespaces sind Verwaltungseinheiten, in denen Klassendefinitionen und Klasseninstanzen eindeutig sind. Sie dienen als Containerobjekte in denen jede Klassendefinition und jede Klasseninstanz eindeutig ist.
Namespaces können sich überschneiden. D.h. ein Objekt kann in mehreren Namespaces auftreten. Die Verwendung von Namespaces erlaubt es, unterschiedliche Sichten auf ein und dieselbe Datenbasis zu haben.

Namespaces können:[4]

- zur Partitionierung von Managementdaten benutzt werden. Dadurch läßt sich in großen Systemen eine Aufteilung der Datenbank erreichen.
- zur Einrichtung von Sichten auf gleichartige Managementobjekte genutzt werden. Es können z.B. alle Router in einem Netzwerk in einem Namespace zusammengefaßt werden.
- zur Vorstrukturierung von Objekten, um z.B. kürzere Abfragezeiten zu erreichen, verwendet werden.

Ein Beispiel für die Verwendung von Namespaces ist bei der Evaluierung des SNMP Providers zu sehen (Kap. 5.1). Hier wird für jedes SNMP-Device ein eigener Namespace angelegt.

3.2 Managed Objects Format (MOF)

Um das beschriebene Modell der Managementinformation möglichst vielen verschiedenen Managementanwendungen zur Verfügung zu stellen wurde eine maschinenlesbare Beschreibungssprache für dieses Modell entwickelt. Diese Sprache wird Managed Objects Format kurz MOF genannt.
Das MOF Format ist von der Interface Definition Language (IDL) abgeleitet.
Unter einer MOF Datei versteht man eine Sammlung von Klassen mit allen Attributen, Referenzen, Methoden sowie Instanzen der definierten Klassen in Textform.
In einer MOF Datei werden Teile eines oder ein gesamtes Schema im Kontext eines Namespace definiert. Kommentare sind innerhalb einer MOF-Datei erlaubt.
Im einzelnen ist die Syntax für die folgenden Objekte definiert:[5]

- Der Syntax einer Klassendefinition läßt sich grafisch so darstellen:

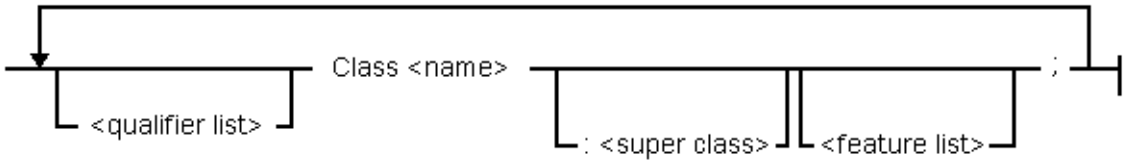

Abbildung 3-3: MOF Klassen Syntax

Eine Klassendefinition wird Optional von einer Liste von Qualifiern (`<qualifier list>`) angeführt. Das Schlüsselwort „`CLASS`“ gefolgt vom Namen der Klasse (`<name>`) markiert den Anfang der eigentlichen Klassendefinition. Ebenfalls optional wird mit der Superklasse (`<super class>`) die Ableitungshierarchie angegeben. Die Liste der Attribute und Methoden (`<feature list>`) wird daran angehängt.

[4] Nach [CIM1, S.40]
[5] Nach [MS1]

- Der Syntax einer Attribut- oder Methodendefinition läßt sich grafisch so darstellen:

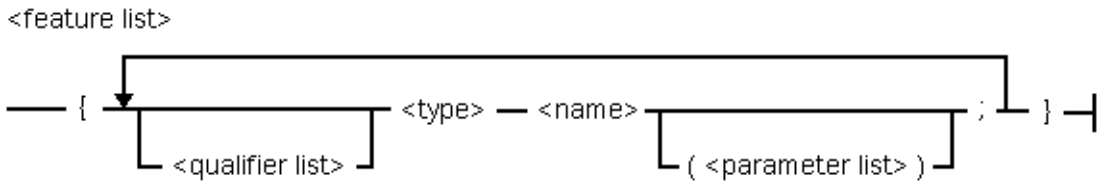

Abbildung 3-4: MOF Attribut Syntax

Eine Attribut- oder Methodenliste ist eine Sequenz von Typ (`<type>`) und Namenskombinationen (`<name>`).
Optional wird diese Definition von einer Qualifierliste (`<qualifier list>`) angeführt. Ebenfalls optional wird eine Parameterliste (`<parameter list>`) angehängt.

- Der Syntax einer Qualifierdefinition läßt sich grafisch so darstellen:

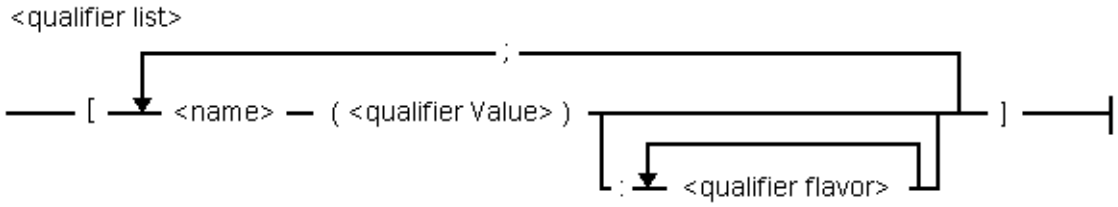

Abbildung 3-5: MOF Qualifier Syntax

Eine Liste von Qualifiern ist eine Sequenz von Kombinationen aus einem Namen (`<name>`) und einem geklammerten Wert für diesen Qualifier (`<qualifier Value>`).
Optional wird diese Kombination von einem Qualifier Flavor (`<qualifier flavor>`) gefolgt. Ein Qualifier Flavor beschreibt das Verhalten eines Qualifiers und hat einen der Werte:

- OVERRIDE
  gibt an das dieser Qualifier in einer Subklasse überschrieben werden kann
- NOOVERRIDE
  gibt an das dieser Qualifier in einer Subklasse nicht überschrieben werden kann
- TOSUBCLASS
  gibt an das dieser Qualifier an eine Subklasse vererbt wird

RESTRICT
gibt an das dieser Qualifier nicht an eine Subklasse vererbt wird

Ein Beispiel für den MOF Syntax ist die Klasse `EventViewerConsumer`:

```
class EventViewerConsumer : __EventConsumer
{
      [key] string Name = "";
      [read] uint8 Severity = 0;
      [read] string
      Description = "";
};
```

Die Klassendefinition beginnt mit dem Schlüsselwort „`class`“ gefolgt vom Namen der Klasse („`EventViewerConsumer`“). Es folgt ein Liste von Attributen mit ihren Typdefinitionen (z.B. „`string NetworkAddress`“). Diese werden von verschiedenen Qualifiern angeführt (z.B. `[read]`)

4 MS-WBEM SDK Architektur

Ein Problem des konventionellen Managements von heterogenen System ist der Zugriff und das Verwalten der einzelnen auf den verschiedenen Systemen liegenden Informationen über die verwalteten Objekte. Meist ist für jedes dieser Systeme eine eigene API oder eine eigene Managementapplikation nötig, um es zu verwalten. WBEM stellt jetzt eine einheitliche Methode zum Zugriff auf diese Daten zur Verfügung.

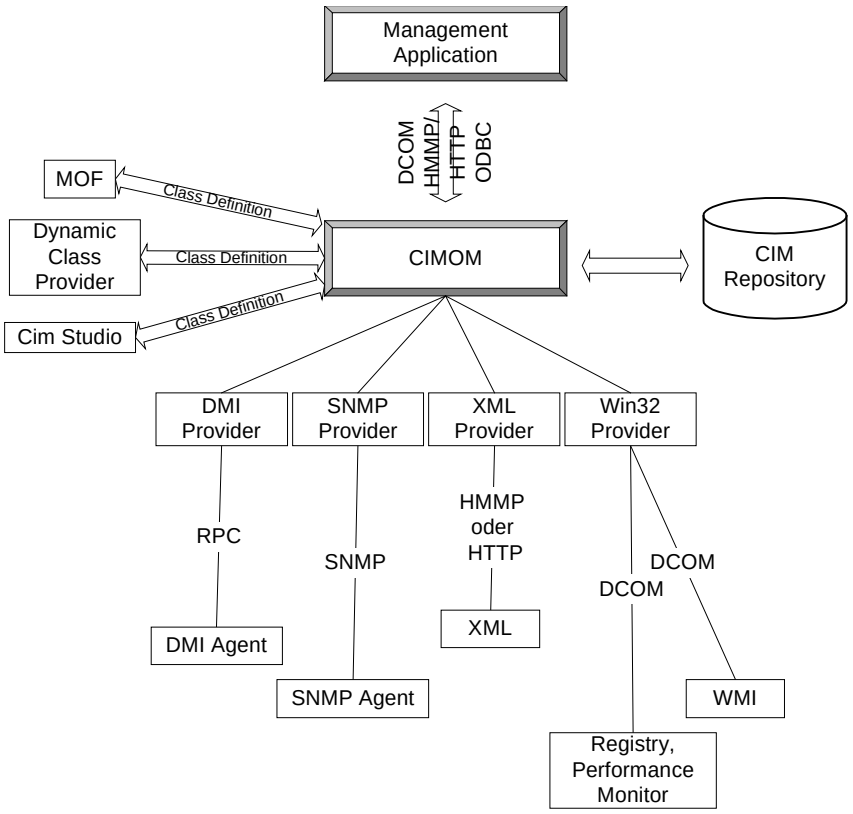

Abbildung 4-1: WBEM Architektur

Wie in Abbildung 4-1 zu sehen ist das zentrale Bindeglied zwischen den einzelnen Komponenten des WBEM SDKs der CIM Objekt Manager (CIMOM). Er verbindet die einzelnen Komponenten und regelt den Zugriff auf das CIM Repository. Wie die meisten Managementarchitekturen basiert auch WBEM auf dem Manager-Agent Prinzip. In der WBEM Architektur wird der Agent jedoch Provider genannt. Die Aufgaben des Managers werden zum Teil vom CIMOM und zum Teil von der Managementanwendung übernommen.

4.1 Komponenten des MS WBEM SDKs

4.1.1 CIM Repository

Das *CIM Repository* ist die Datenbank des MS WBEM SDKs. Es enthält statische Klassendefinitionen und statische Klasseninstanzen. Sämtliche zur Verwaltung des MS WBEM SDKs nötige Information (z.B. Security Information, Information über die Fähigkeiten und die Aufrufschnittstellen von Providern) ist ebenfalls über Klassendefinitionen und Klasseninstanzen realisiert und wird im CIM Repository gespeichert.
Änderungen an statischen Klassendefinitionen und -instanzen können über eine Schnittstelle im CIMOM oder durch Compilierung von MOF Dateien mit einem speziellen MOF Compiler durchgeführt werden.

Das CIM Repository ist als Datei verwirklicht. Es befindet sich standardmäßig in dem Verzeichnis „%Systemroot%\SYSTEM32\WBEM\Repository“.

4.1.2 CIMOM

Der *CIM Object Manager (CIMOM)* ist die zentrale Bindeglied in der WBEM Architektur. Der CIMOM regelt den Zugriff auf das CIM Repository. Er stellt dabei ein Interface zum Zugriff auf das CIM Repository bereit. Über dieses Interface können im CIM Repository einerseits Klassendefinitionen und statische Klasseninstanzen hinzugefügt, überschrieben oder gelöscht werden.
Andererseits können Management Applikationen Informationen über die Managementobjekte abfragen.
Beim Abfragen von Informationen über Managementobjekte werden statische Klasseninstanzen, die in das CIM Repository importiert wurden, oder dynamische Klasseninstanzen die von einem Provider geliefert werden an die Managementapplikation zurückgeliefert. Zum Erlangen der dynamischen Klasseninstanzen wird nach einem Eintrag für den die Klasse betreffenden Provider gesucht und dieser dann aufgerufen. Der Provider liefert die dynamischen Klasseninstanzen an den CIMOM zurück, der sie wiederum an die Managementanwendungen weitergibt.

Der CIMOM verwaltet zudem die Zugriffsrechte, sowohl auf das Repository als auch auf die dynamischen Informationen. Das MS WBEM SDK verwaltet allerdings nicht wie im WBEM Standard vorgesehen unabhängige Zugriffsrechte auf einzelne Objekte sondern sichert den Zugriff nach dem Motto „ganz oder gar nicht“. Für einzelne Benutzer oder Gruppen kann die Sicherheit nur auf alle vom CIMOM zugreifbaren Objekte einheitlich geregelt werden. Eine Abstufung ist nur auf der Ebene

- schreibender Zugriff
- lesender Zugriff
- keine Zugriff

möglich.

Softwaretechnisch ist der CIMOM als Service realisiert, der automatisch gestartet wird.

4.1.3 Provider

WBEM, wie oben schon erwähnt, basiert wie fast alle „klassischen“ Managementarchitekturen auf dem Manager-Agent Prinzip. Der Provider übernimmt dabei die Rolle des Agenten und liefert sämtliche Managementinformationen an den CIMOM.
Dabei wird zwischen drei Arten von Providern unterschieden:

- Dynamic Class Providern
- Dynamic Instance Providern
- Dynamic Property Providern

Dynamic Class Provider haben die Aufgabe, Klassendefinitionen dynamisch zu erzeugen. Diese werden nicht im Repository abgelegt. CIMOM speichert sie statt dessen selbst, die Lebenszeit ist daher auf die Laufzeit beschränkt.
Dynamische Klassen sind durch die begrenzte Lebenszeit flexibler in der Handhabung als statische.

Dynamic Instance Provider füllen statische und dynamische Klassen mit Instanzen. Während statische Klassen auch statische Instanzen haben können, besitzen dynamische Klassen ausschließlich dynamische Instanzen. Diese dynamischen Instanzen werden wie die dynamischen Klassen auch nur im CIMOM gespeichert, nicht im Repository.
Der Provider für dynamische Instanzen wird stets im Qualifier-Bereich der Klassendefinition angegeben.
Dynamic Instance Provider können Informationen der Managed Objects lesen und schreiben sowie Operationen auf diesen ausführen wenn die entsprechenden Funktionen im Provider implementiert wurden.

Dynamic Property Provider belegen dynamisch Attribute von statischen Klasseninstanzen mit Werten und lesen diese aus.
Der Provider wird hier in der Attributdefinition angegeben. Dadurch ist es möglich, einen Teil der Attribute statisch zu belegen und einen Teil dynamisch.

4.1.4 Sonstige Komponenten

Mit dem MS WBEM SDK werden noch weitere Komponenten mitgeliefert:
Der MOF Compiler, das WBEM Developer Studio und der MS WBEM Event Viewer. Letzterer wird in Kapitel 5.3 behandelt, daher wird an dieser Stelle nur auf die beiden ersten eingegangen.

Der MOF Compiler überprüft die Syntax von MOF Files und trägt die in MOF Files vorhandenen Informationen in das CIM Repository ein. Sollte dies nicht möglich sein, liefert er eine Fehlermeldung zurück. Dabei ist zu beachten, daß er kein Rollback durchführt, d.h. Informationen die ins Repository übertragen wurden bevor der Fehler aufgetreten ist, bleiben in diesem gespeichert.

Das WBEM Developer Studio stellt eine beispielhafte Managementanwendung dar. Es bietet die Möglichkeit, Klassendefinitionen, Klasseninstanzen und Attribute zu erzeugen und zu manipulieren. Auch Verknüpfungen können graphisch dargestellt werden (siehe Kapitel 6.1).

4.1.5 Mitgelieferte Provider

Um WBEM als ein offenes möglichst einfach zu implementierendes Mangementframework zu platzieren, sollen verschiedene Provider schon in der Version 1.0 des SDKs mitgeliefert werden. Neben Providern, die die Integration von vorhandenen und verbreiteten Managementmodellen wie SNMP und DMI sichern, liefert Microsoft ebenfalls Provider, die den Zugriff auf systemspezifische Komponenten von Windows NT wie die Registry und den Performance Monitor zur Verfügung stellen. Folgende Provider sind bereits im SDK integriert:

- SNMP:
  Der SNMP Provider stellt den Zugriff auf beliebige lokale oder entfernte SNMP Agenten zur Verfügung. Es ist sowohl lesender als auch schreibender Zugriff möglich.
- DMI:
  Der DMI Provider stellt den Zugriff auf beliebige lokale oder entfernte DMI Agenten zur Verfügung. Es ist sowohl schreibender und lesender Zugriff wie auch die Ausführung von Methoden möglich.

- NT spezifische WIN32 Provider:
  Diese Provider stellen den Zugriff auf NT spezifische Komponenten zur Verfügung. Dazu zählen: Registry, Event Log, Performance Monitor, Informationen über die laufenden Prozesse. Es ist sowohl schreibender und lesender Zugriff wie evtl. die Ausführung von Methoden möglich (Methoden sind allerdings nur marginal im Build 486 (beta2) implementiert).

4.2 Verwaltungsinformationen

Alle zur Verwaltung des Systems notwendigen Informationen sind ebenfalls als CIM Klassen modelliert. Dieses Vorgehen erinnert an den Aufbau von SQL Datenbanken. Solche Verwaltungsklassen sog. System Klassen können am Präfix ‚__' (zwei Unterstriche) erkannt werden.

Providerregistrierung

Die Registrierung von Providern erfolgt als Instanz einer von der Klasse „__Provider" abgeleiteten Klasse. Ein Event-Provider wird z.B. als Instanz der Klasse „__EventProvider" ein Instanz-Provider als Instanz der Klasse „__InstanceProvider" modelliert.

4.3 WQL als Abfragesprache

Eine Möglichkeit gezielt Abfragen aus einer Managementanwendung heraus auf Instanzen von Klassen zu machen ist im MS WBEM SDK die sog. WBEM Query Language (WQL). Diese SQL ähnliche Abfragesprache erlaubt es Auswahl („SELECT") Abfragen auf Klasseninstanzen zu machen. Die Ergebnismenge wird dabei durch die Angabe des von SQL bekannten Schlüsselwortes „WHERE" gefolgt von einer Anzahl von Ausdrücken eingeschränkt.

Ein Beispiel für eine WQL Abfrage:

```
select * from RegistryTreeChangeEvent
where Hive = "HKEY_LOCAL_MACHINE" and RootPath = "Software\Fopra1"
```

In dieser Abfrage wird jede Instanz der Klasse `RegistryTreeChangeEvent` ausgewählt, das folgenden Kriterien entspricht:

- das Attribut „`Hive`“ der Klasse hat den Wert `HKEY_LOCAL_MACHINE`
- das Attribut „`RootPath`“ der Klasse hat den Wert `Software\Fopra1`

In WQL können allerdings nur Informationen gesammelt werden, schreibende Zugriffe wie beispielsweise „Insert“, „Update“ und „Delete“ sind nicht möglich.
Diese Abfragesprache wird bei der Behandlung von Events verwendet (Kapitel 5.3).

5 Evaluierte Provider

Ziel der Arbeit war es unter anderem die Integration von verschiedenen vorhandenen Managementarchitekturen in diese Version des MS WBEM SDKs zu testen. Jeder dieser Architekturen wird über einen eigenen Provider angesprochen. Die ersten beiden Kapitel 5.1 und 5.2 zeigen die Integration der Managementarchitekturen SNMP und DMI. In den folgenden Kapiteln werden weitere Aspekte wie Events, Assoziations Klassen und der im MS WBEM SDK enthaltene WIN32 Provider behandelt.

Problematisch bei der Evaluierung war der Versionswechsel während der Evaluierungsphase. Durch die rapide Entwicklung war die Dokumentation z.T. nicht mit der Implementierung konsistent. Teilweise sind in der Dokumentation Klassen und Attribute beschrieben die nicht mehr Bestandteil des CIM Schemas sind. Besonders problematisch war dies bei der Evaluierung des SNMP Providers.

5.1 SNMP

Des Thema SNMP stellt bei der Evaluierung des Microsoft WBEM SDKs ein spezielles Problem dar. Die konkrete Implementierung der SNMP Provider hat sich durch den Versionswechsel stark verändert. Da allerdings die Dokumentation auf diese neue Implementierung nicht angepaßt wurde, konnte die Integration von SNMP in das MS WBEM SDK nur in der ersten Version, Beta2 Build 220, getestet werden.

5.1.1 Beschreibung der Architektur

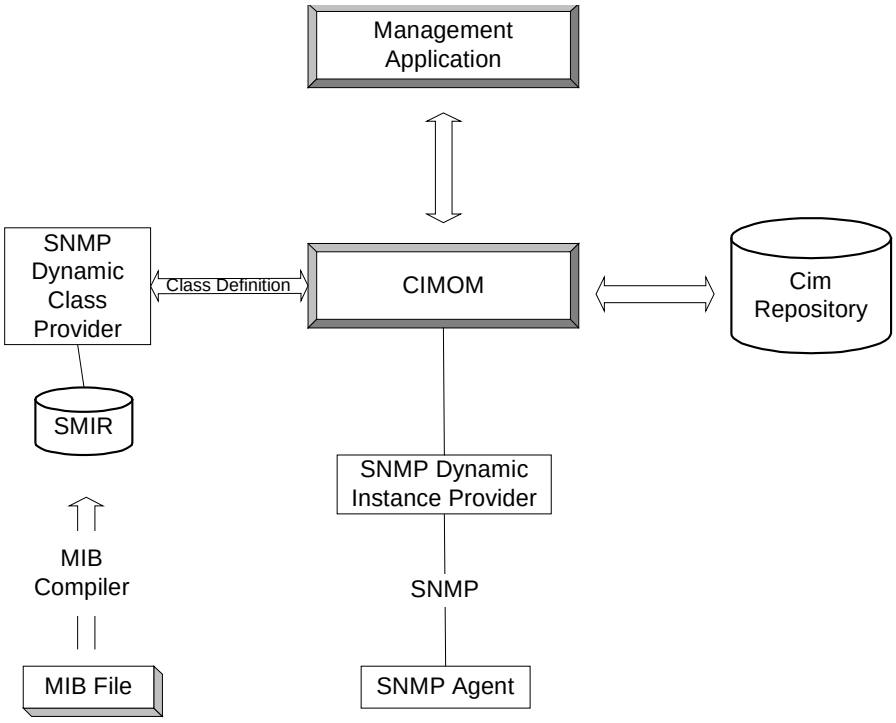

Abbildung 5-1: SNMP - Architektur

Die Integration der SNMP Management Architektur in das MS WBEM SDK erfolgt relativ kompliziert über verschiedene Komponenten:

- Ein Compiler, der MIB Files in sog. SMIR Files umwandelt. Diese SMIR Files können direkt mit dem MOF Compiler im CIM Repository Klassendefinitionen anlegen oder dienen als Eingabe für den „SNMP Dynamic Class Provider“.
- Einen „Dynamic Class Provider“ der aus vorher umgewandelten MIB Files (SMIR) entsprechende Klassendefinitionen im CIM Repository generiert. Der Dynamic Class Provider ließt dazu alle im SMIR Repository eingetragenen SMIR Dateien aus und generiert entsprechende Klassendefinitionen im CIM Repository. Das SMIR Repository ist dabei als eine Sammlung von Dateien verwirklicht.
- Einen „Dynamic Instance Provider“ der die SNMP Kommunikation mit einem SNMP Agenten abwickelt und die entsprechenden Klasseninstanzen erzeugt. Es können jeweils nur Instanzen von vorher ins SMIR Format übersetzten und eingelesenen MIB Dateien erzeugt werden.

Änderungen gegenüber Build 220 in Build 468

Wie schon eingangs beschrieben, haben sich im Build 468 gravierende Änderungen ergeben. Dies hat sich sogar bis in die Architektur ausgewirkt. So ist zu den schon aus Build 220 bekannten Providern ein SNMP Event Provider hinzugekommen. Dieser Provider ist für die Umsetzung von SNMP Traps in WBEM Events verantwortlich.

5.1.2 Implementierung

Beschreibung der Anforderungen

Es soll auf einer beliebigen SNMP-fähigen Komponente das Lesen und Setzen von SNMP Variablen und Tabellen gezeigt werden. Exemplarisch wird dazu die MIB Variable SysContact verwendet.

Beschreibung der Lösung

Zum Test des SNMP Providers wird auf einen entfernten, SNMP fähigen Rechner lesend und schreibend zugegriffen. Der Zugriff erfolgt über ein TCP/IP Netzwerk zwischen dem MS WBEM SDK Rechner und dem zweiten SNMP fähigen Rechner.

Folgendes Vorgehen ist zum Installieren des SNMP Providers nötig:

1. Anlegen einer Instanz der __Namespace Klasse
2. Registrieren des SNMP Providers im angelegten Namespace
3. Compilieren eines SMIR Files um die Klassendefinitionen für die entsprechende SNMP Variable zu erzeugen

Konkret ist dieses Verfahren in der folgenden MOF Datei verwirklicht.

```
//**************************************************************************
//* Instances of: __NAMESPACE
//* with qualifiers AgentAdress, AgentReadCommunityName,
AgentWriteCommunityName
//* The qualifers identify the host on wich the get/set commands should
be executed
//**************************************************************************
[AgentAddress ("10.1.1.1"), AgentReadCommunityName ("public"),
AgentWriteCommunityName ("public")]
instance of __NAMESPACE
{
      Name = "SNMP";

};
```

Zuerst wird eine neue Instanz der __Namespace Klasse mit den Schlüsselwörtern `instance of __NAMESPACE` erzeugt. Das einzige Attribut (`Name`) der Klasse wird mit dem Wert „SNMP" gefüllt.
Wichtig sind bei dieser Klasseninstanziierung vor allem folgende Qualifier:

- AgentAdress: Adresse des Agenten (z.B. IP Adresse, DNS Name, Netbios Name) im Beispiel: `10.1.1.1`
- AgentReadCommunityName: Community String für lesenden Zugriff auf den SNMP Agenten. Im Beispiel: `public`
- AgentWriteCommunityName: Community String für schreibenden Zugriff auf den SNMP Agenten. Im Beispiel: `public`

```
//**************************************************************************
//* Instances of: __WIN32PROVIDER
//* this registers the SNMP instance provider for the specific namespace
//**************************************************************************

#pragma namespace("\\\\.\\ROOT\\DEFAULT\\SNMP")

instance of __WIN32PROVIDER
{
    Provider = "MS_SNMP_INSTANCE_PROVIDER";
    ProviderClsId = "{D45BD3C1-EF23-11cf-8CBC-00AA00A4086C}";
    MethodSet = 331793;
} ;
```

Mit dem Befehl `#pragma namespace("\\\\.\\ROOT\\DEFAULT\\SNMP")` werden sämtliche folgenden Klassendefinitionen und Klasseninstanzen im neu angelegten Namespace („ROOT\\DEFAULT\\SNMP") erstellt.
Mit den Schlüsselwörtern `instance of __WIN32PROVIDER` wird der SNMP Instance Provider registriert.
Die Attribute `ProviderClsId` und `MethodSet` sind für diese Providerimplementierung spezifische Werte. Das Attribut `Provider` enthält den Namen des Providers.

Anschließend wird die im MS WBEM SDK mitgelieferte Datei „RFC1213.mof“ mit Hilfe des MOF Compilers in das CIM Repository im neu angelegten Namespace kompiliert. Diese Datei enthält die im RFC1213 definierten MIB Variablen in übersetzter Form. Dadurch werden im CIM Repository Klassendefinitionen mit den MIB Variablen als Attribute erzeugt. Der SNMP Instance Provider instanziiert diese Klassen und die Managementinformation kann aus den Klasseninstanzen ausgelesen werden.
Diese Datei wurde durch Übersetzung der entsprechenden MIB Datei mit dem in Kapitel 5.1.1 beschriebenen MIB-SMIR Compiler erzeugt.

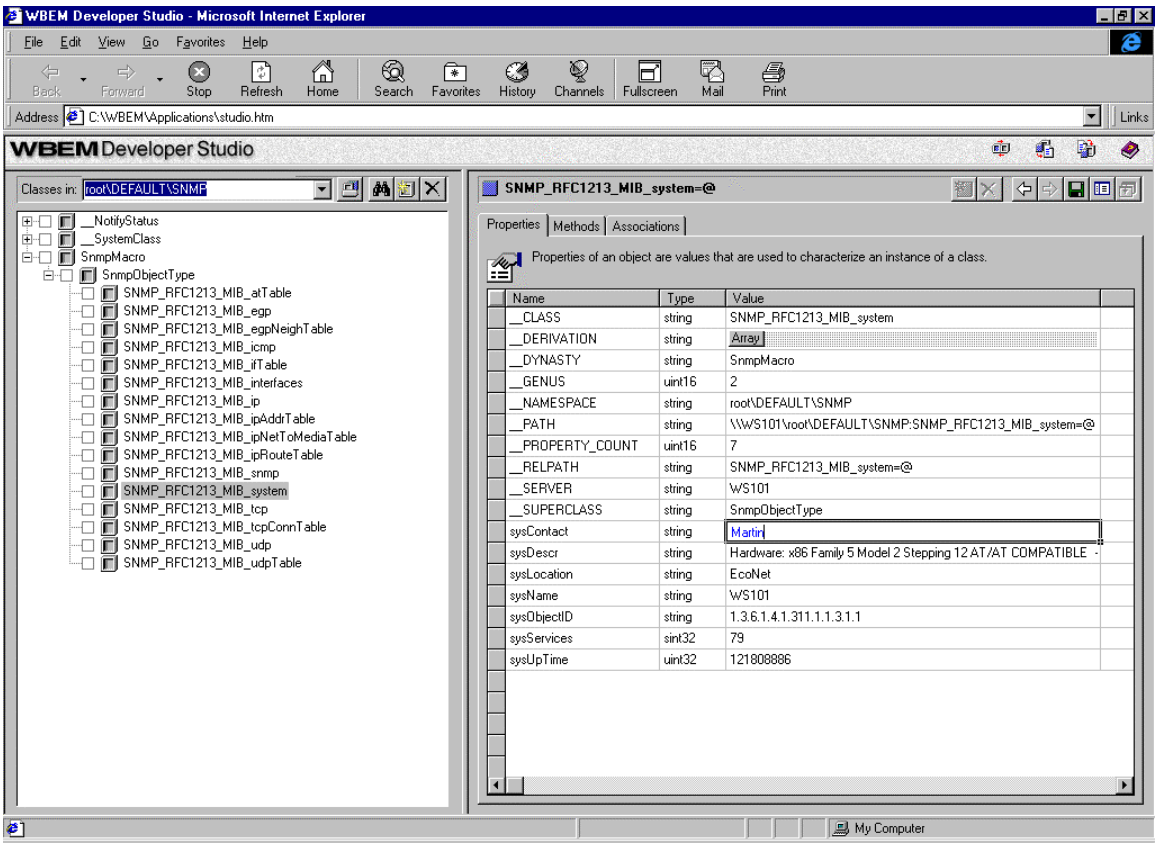

Abbildung 5-2: MS WBEM SDK Dev. Studio mit SNMP Variable sysContact

Abbildung 5-2 zeigt die MIB Variable „SysContact“ im Developer Studio als Attribut der Instanz der Klasse „SNMP_RFC1213_MIB_system“.

Das Auslesen von Tabellen erfolgt auf identisch mit dem Auslesen einer einzelnen Variable. Der einzige Unterschied besteht nur in der Klassendefinitionen für die zu lesende Tabelle. Diese Klassendefinition wird durch den in Kapitel 5.1.1 beschriebenen MIB-SMIR Compiler erzeugt.
Als Beispiel sei hier die Tabelle „SNMP_RFC1213_MIB_ipRoutTable“ aus Abbildung 5-2 genannt.

Probleme und Hindernisse bei der Implementierung

- Die im MS WBEM SDK mitgelieferte Dokumentation ist äußerst ungenau und enthält keinerlei Beispiele. Die Folge davon ist langwieriges Ausprobieren.

5.2 DMI

5.2.1 Beschreibung der Architektur

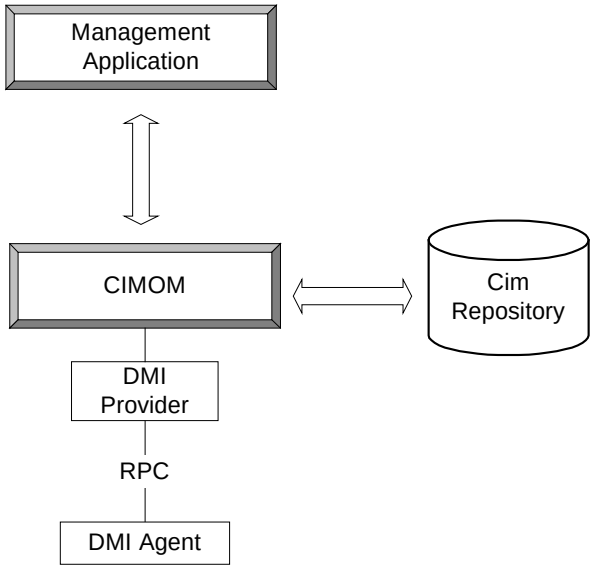

Abbildung 5-3: DMI - Architektur

Die Integration der DMI Managementarchitektur in das MS WBEM SDK erfolgt über verschiedene Provider die den Zugriff auf die gesamte Funktionalität von DMI sicherstellen.
Im einzelnen sind dies folgende Provider:

- Dynamic Class Provider
  dieser Provider erzeugt je nach Fähigkeiten des DMI Agenten entsprechende Klassendefinitionen für die DMI Information
- Dynamic Instance Provider
  erzeugt Klasseninstanzen die die DMI Information in WBEM abbilden
- Method Provider
  bildet Funktionen die der DMI Agent auf dem MO ausführen kann auf WBEM Methoden ab
- Event Provider
  bildet Ereignisse die vom DMI Agenten erzeugt werden auf WBEM Events ab

Softwaretechnisch ist dieser Provider als eine integrierte Komponente verwirklicht.

5.2.2 Implementierung

Beschreibung der Anforderungen

Es sollen von einem entfernten DMI Agenten beliebige Daten ausgelesen werden. Dazu muß zu dem entfernten Agenten eine Netzverbindung vorhanden sein über die sog. Remote Procedure Calls (RPCs) ausgeführt werden können.
Die Abbildung von DMI Ereignissen auf WBEM Events wird in Kapitel 5.3 behandelt.

Beschreibung der Lösung

Folgendes Vorgehen ist für die Integration von DMI in das MS WBEM SDK vorgesehen:

1. Anlegen eins neuen Namespace für jeden DMI Agenten (=jede DMI fähige Komponente im Netzwerk hat ihren eigenen Namespace
2. Definieren der Klasse „DmiNode“ im neuen Namespace
3. Instanziieren der neuen Klasse „DmiNode“ im neuen Namespace mit der Netzwerkadresse des Agenten
4. Registrierung der Provider

Im einzelnen wird dieses Verfahren in der folgenden MOF Datei verwirklicht.

```
//*********************************************************************
//* Instances of: __NameSpace
//* one Instance per DMI Node
//*********************************************************************
instance of __NameSpace
{
      Name = "DMINODE1";
};
```

Zuerst wird eine neue Instanz der __Namespace mit den Schlüsselwörtern `instance of __NAMESPACE` erzeugt. Das einzige Attribut (`Name`) der Klasse wird mit dem Wert „`DMINODE1`” gefüllt.

```
//*************************************************************************
//* Class DmiNode for registering the DMI Agent with its Network Adress
//*
//*************************************************************************
#pragma namespace("\\\\.\\root\\DMINODE1")

[singleton]
class DmiNode
{
      string NetworkAddress;
};
```

Mit dem Befehl `#pragma namespace("\\\\.\\root\\DMINODE1")` wird angegeben das alle folgenden Klassendefinitionen und Klasseninstanzen im neu angelegten Namespace erzeugt werden.
Anschließend wird eine Klasse „DmiNode“ definiert werden die als einziges Attribut die Netzadresse eines DMI Agenten hat. Der Qualifier `[singelton]` gibt an, daß es von dieser Klasse nur eine Instanz geben kann.
Das einzige Attribut der Klasse („`string NetworkAddress`“) gibt in der Instanz die Adresse des DMI Agenten an.

```
//*************************************************************************
//* Instances of: DMI Node
//* with this instance the DMI Agent 10.1.1.1 is registered
//*************************************************************************
instance of DmiNode
{
      NetworkAddress = "10.1.1.1";
};
```

Für den DMI Agenten wird eine statische Instanz der neuen Klasse DmiNode erzeugt. Das Attribut „`NetworkAddress`“ gibt in diesem Beispiel die Adresse des DMI Agenten mit „`10.1.1.1`” an.

Im neuen Namespace müssen der Dynamic Class Provider und der Dynamic Instance Provider registriert werden.

```
//*************************************************************************
//* Registration of the DMI Providers
//*
//*************************************************************************
instance of __Win32Provider As $Provider
{
    Name           = "WbemDmip" ;
    ClsId          =  "{DE065A70-19B5-11D1-B30C-00609778D668}" ;
};
```

Zuerst wird eine Instanz der Klasse „__Win32Provider“ mit den Schlüsselworten „`instance of __Win32Provider As $Provider`” erzeugt. Das Schlüsselwort „`$Provider`“ wird zur späteren Referenzierung der erzeugten Klasse gebraucht.

```
instance of __InstanceProviderRegistration
{
```

```
    Provider = $Provider;
    SupportsGet = TRUE;
    SupportsPut = TRUE;
    SupportsDelete = TRUE;
    SupportsEnumeration = TRUE;
};
```

Anschließend wird eine Instanz der Klasse „__InstanceProviderRegistration“ mit den Schlüsselworten „`instance of __InstanceProviderRegistration`” erzeugt.
Das Attribut „Provider“ referenziert die oben erzeugte Instanz des WIN32 Providers. Die restlichen Attribute übergeben Informationen über die Fähigkeiten des Providers.

```
instance of __ClassProviderRegistration
{
    Provider = $Provider;
    SupportsGet = TRUE;
    SupportsPut = FALSE;
    SupportsDelete = TRUE;
    SupportsEnumeration = TRUE;
    QuerySupportLevels = NULL ;
      ResultSetQueries = {
        "Select * From meta_class Where __this isa \"DmiComponent\"",
        "Select * From meta_class Where __this isa \"DmiGroupRoot\"" ,
        "Select * From meta_class Where __this isa \"DmiBindingRoot\"" ,
        "Select * From meta_class Where __this isa \"DmiNodeData\"" ,
        "Select * From meta_class Where __this isa \"DmiLanguage\"" ,
        "Select * From meta_class Where __this isa \"DmiEvent\"" ,
        "Select * From meta_class Where __this isa
\"DmiAddMethodParams\"" ,
        "Select * From meta_class Where __this isa \"DmiGetEnumParams\""
,
        "Select * From meta_class Where __this isa
\"DmiLanguageMethodsParams\""
      } ;
} ;
```

Die Registrierung des Class Providers erfolgt äquivalent zur Registrierung des Instance Providers. Auffällig ist hier nur das Attribut „`ResultSetQueries`“.
Dieses Attribut enthält eine Liste von WQL Statements. Diese WQL Statements erzeugen eine Liste von Klassen die vom Class Provider erzeugt werden können. Dem CIMOM werden über dieses Attribut die Namen der Klassen die vom Class Provider erzeugt werden können mitgeteilt.

Mit einer Management Applikation oder dem im MS WBEM SDK integrierten WBEM CIM Studio ist es jetzt möglich die DMI Informationen auszulesen.

Probleme und Hindernisse bei der Implementierung

- Durch die relativ gute Dokumentation und die Beispiele war es vergleichsweise einfach hier ein brauchbares Ergebnis zustande zu bringen.

5.3 Events

5.3.1 Beschreibung der Architektur

Definition

Im MS WBEM SDK sind WBEM-Events Ereignisse die entweder mit Bedingungen der realen Umgebung oder mit Veränderungen des Repository des CIMOM zusammenhängen. Der CIMOM steuert dabei die Kommunikation zwischen den beteiligten Komponenten. Für diese ist die Existenz der jeweils anderen transparent. Hierbei wird zwischen *internen (intrinsic)* und *externen (extrinsic)* Events unterschieden: Die internen Events betreffen alle Veränderungen des Repositorys. Es wird jeweils bei *Erzeugung/Veränderung/Löschen (Creation/Modification/DeletionEvent)* von Klassen, Instanzen und Namespaces ein entsprechender Event ausgelöst. Alle anderen Ereignisse werden als externe Events klassifiziert. Im Gegensatz zu den genau festgelegten internen Events kann es eine Vielzahl von externen Events geben. Hier ist es Sache der Provider, die jeweils benötigten Events zu melden.
Darunter fallen beispielsweise Veränderungen der Registry oder ähnliches.
Event wird dabei nur das auslösende Ereignis selbst genannt, die Meldungen von Events werden mit *Notification* bezeichnet.

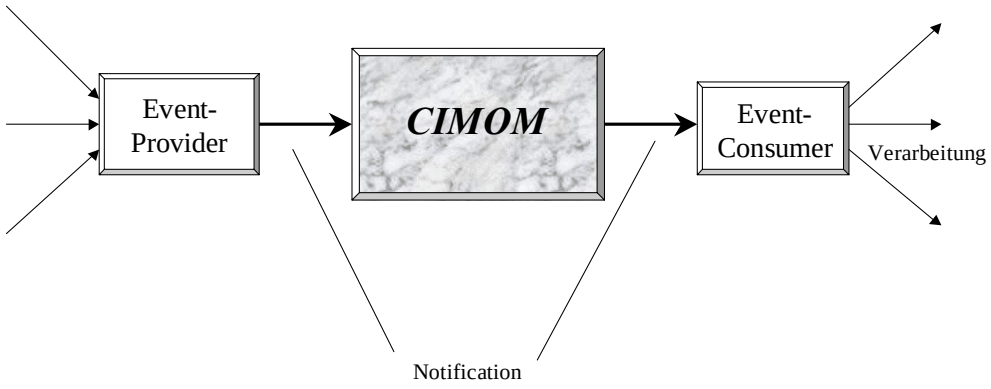

Abbildung 5-4: Notifications

Event-Modell

Die folgende Abbildung verdeutlicht den Ablauf von Notifications (in diesem Fall bei permanenten Consumern, bei temporären Consumern fällt der Kasten *CIMOM Repository* weg, da die Abfrage direkt im logischen Consumer mitgespeichert wird).
Auf ein Ereignis hin schickt ein Event-Provider eine Notification an den CIMOM. Der überprüft, ob ein Event-Filter existiert, der von diesem Event betroffen ist. Sollte dies der Fall sein, sucht er den entsprechenden logischen Consumer heraus und ermittelt den zugehörigen Event-Consumer-Provider. Durch diesen (der ggf. neu gestartet werden muß) wird die Verbindung zum Sink hergestellt und diesem dann die Notification übermittelt.

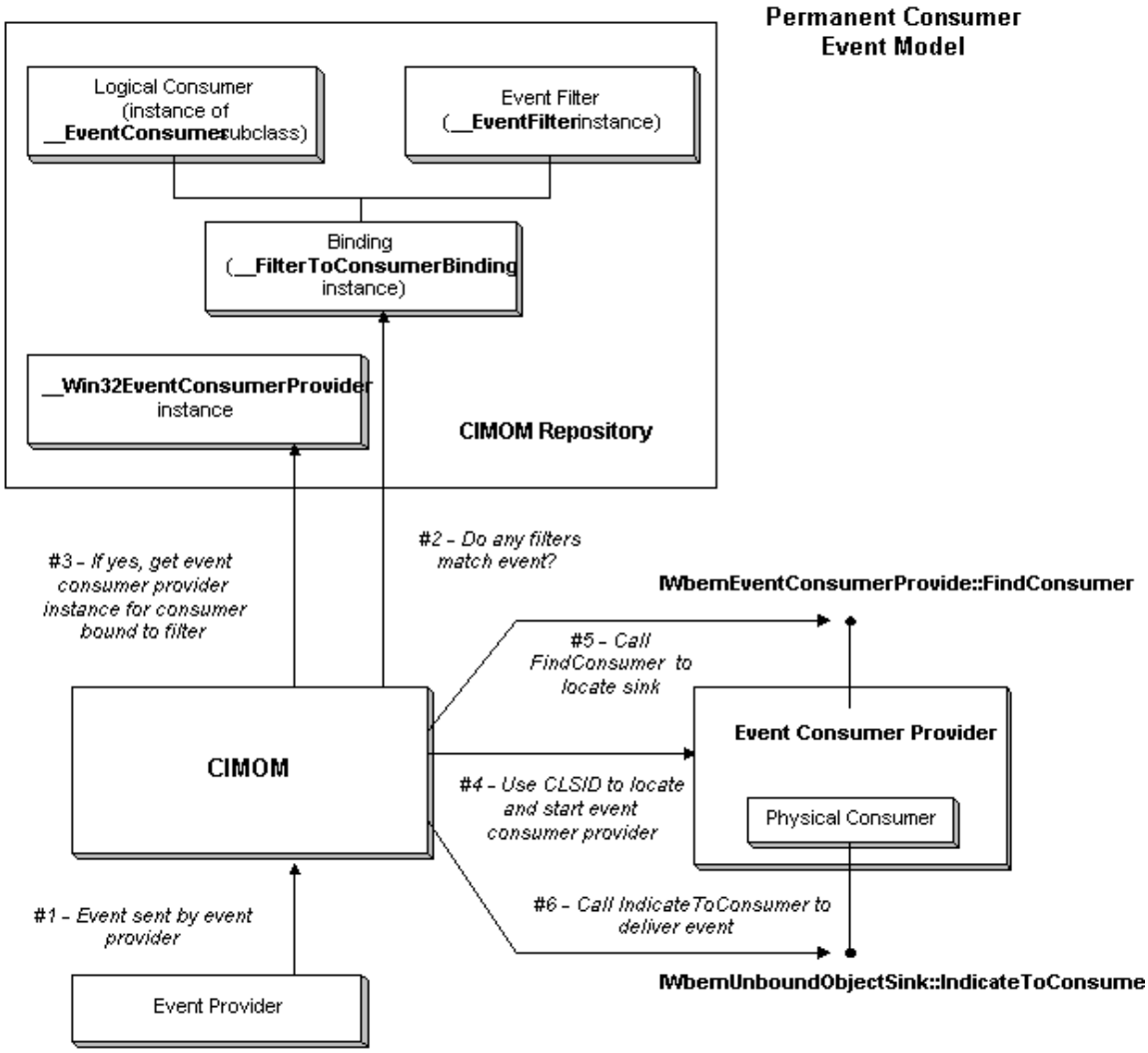

Abbildung 5-5: Event Modell

Beteiligte Komponenten

Event-Provider:

Event-Provider sind dafür zuständig, daß auftretende *Events* als sogenannte *Notifications* dem CIMOM mitgeteilt werden. Dabei spielt es für den Provider keine Rolle, was anschließend mit diesen Notifications geschieht.
Sowohl CIMOM als auch Event-Provider können Notifications für *interne* Events senden. CIMOM liefert die Notifications für Änderungen an *statischen*, während die Event-Provider die Änderungen an den *dynamischen* Klassen und Instanzen (Namespaces sind ein Sonderfall von Klassen) melden.
Externe Events werden stets von Providern gemeldet, nachdem diese die Ereignismeldung von der entsprechenden Quelle erhalten haben. Quellen können sowohl Hardwarekomponenten wie Festplatten, Lüfter, Temperaturfühler, Türschlösser oder ähnliches als auch Softwarekomponenten wie beispielsweise Datenbanken sein.

Event-Consumer:

Event-Consumer sind die Empfänger der Notifications von Events. Consumer geben über einen Abfrage-String an, über welche Events sie benachrichtigt werden sollen.
Es gibt *temporäre* und *permanente* Consumer.
Temporäre Consumer empfangen Notifications nur wenn sie gerade laufen. Wenn sie inaktiv sind, werden die Notifications auch nicht gespeichert. Typische temporäre Consumer sind Anwendungen, die bestimmte Daten anzeigen und Notifications über Events benötigen, die eben diese Daten betreffen. Wenn so eine Anwendung nicht läuft, benötigt sie auch die entsprechenden Notifications nicht.
Permanente Consumer müssen nicht aktiv sein, wenn die Notification beim CIMOM ankommt. Hier überprüft dieser zunächst, ob der Consumer läuft und startet ihn gegebenenfalls. In der Zwischenzeit werden die Notifications gespeichert.
Permanente Consumer sind COM-Objekte und laufen als In-Process-dll, Lokale Server, oder Remote Server (die beiden letzteren erfordern Distributed COM (DCOM) auf der Plattform, auf der sie laufen). Sie bieten die Möglichkeit, Notifications asynchron oder synchron zu empfangen. Bei der synchronen Zustellung wartet der Provider bis die Notification von dem entsprechenden Consumer vollständig empfangen wurde, bei der asynchronen Zustellung nicht.
Permanente Consumer bauen auf einer mehrstufigen Architektur auf: „Ansprechpartner“ für den CIMOM ist ein *logischer* Consumer. Dieser ist mit einer Instanz der Win32Provider (einem Event-Consumer-Provider) verknüpft, die wiederum auf ein COM-Objekt zeigt (den physikalischen Provider), das dann den Zugang zu dem physischen Consumer (das oben genannte COM-Objekt), der auch als *Sink* bezeichnet wird, herstellt. Die beiden COM-Objekte können sich dabei durchaus in *einer* Anwendung befinden. Wichtig ist hierbei nur, daß über die beiden Schnittstellen (IWbemEventConsumerProvider, IWbemUnboundObjektSink) der physische Consumer vom logischen aus erreicht wird.

Event-Consumer-Provider

Event-Consumer-Provider dienen dazu, die Verbindung zwischen dem CIMOM und dem physischen Consumer herzustellen.
Zu einem Event-Consumer-Provider gehört zum einen ein COM-Objekt (der Provider selbst) sowie ein oder mehrere physische Consumer (*Sinks*).
Wird ein logischer Consumer zum ersten Mal benützt, so ruft CIMOM die Funktion *FindConsumer* des entsprechenden Event-Consumer-Providers auf, der dann die Verbindung zu dem jeweiligen Sink herstellt. Weitere Events werden direkt zu diesem Sink gesendet. Erfolgt während einer bestimmten (konfigurierbaren) Zeit keine Notification (für diesen Consumer), so wird er freigegeben. Danach muß er bei einem erneuten Aufruf wieder von dem Provider gesucht werden (s.o.).

Sink (Consumer COM-Objekt)

Das physische Objekt, das den Consumer darstellt, wird mit *Sink* bezeichnet.
Es gibt mehrere Möglichkeiten, Sinks zu logischen Consumern zuzuordnen:

- Einen Sink für alle logischen Consumer
- Je einen Sink für jeden logischen Consumer
- Je einen Sink für jede Gruppe von logischen Consumern

Nach welcher Strategie implementiert wird, hängt von den Anforderungen ab. So ist die erste Möglichkeit auf Platzbedarf (nur ein COM-Objekt im Speicher), die zweite auf

Performanz (es stehen für bestimmte Events die jeweiligen Consumer bereit) hin optimiert. Die dritte stellt einen Kompromiß aus beiden dar.

Event-Filter:

Der Event-Filter beinhaltet eine Abfrage, die eine Auswahl aus Events liefert. Hierzu sollten natürlich nur die wirklich benötigten Events aufgenommen werden. Momentan unterstützt WBEM nur *WQL* als Abfrage-Sprache, daher muß diese auch angegeben und benützt werden. Prinzipiell ist allerdings die Verwendung anderer Abfrage-Sprachen möglich.
Die *WBEM Query Language* (WQL) ist ein Abkömmling von SQL, die um WBEM-spezifische Dinge erweitert wurde. Erlaubt sind aber im Gegensatz zu SQL nur Lese-Vorgänge (kein Update, Insert oder Delete).
Die Event-Filter stellen die Brücke zwischen logischen Consumern und Event-Providern dar. Im Attribut *Query* wird auf bestimmte (definierte) Events bezug genommen, die von einem (bestimmten) Provider „bedient“ werden. In Instanzen der Klasse *__FilterToConsumerBinding* werden dann Filtern entsprechende logische Consumer zugeordnet.

Registrierung

Um Events auswerten zu können, sind einige Registrierungen im CIMOM Repository notwendig. Folgende Komponenten müssen dabei registriert werden:

- Events
- Event Provider
- Consumer
- Event Consumer Provider
- Event Filter
- Verknüpfung zwischen Filter und Consumer

Im Einzelnen sind folgende Schritte notwendig:

- Die Events, die genutzt werden sollen, müssen als Klassen definiert werden (Unterklassen von __Event (die internen Klassen) bzw. von __ExtrinsicEvent:__Event).
- Der Event Provider muß registriert werden, einmal als Instanz einer Unterklasse von __Provider und andererseits als Instanz der Klasse __EventProviderRegistration, die Unterklasse von __ProviderRegistration ist.
- Der logische Consumer muß registriert werden (Instanz einer Unterklasse von __EventConsumer).
- Im Fall von temporären Consumern genügt es, diese zu registrieren. Bei permanenten Consumern muß zusätzlich ein Event-Consumer-Provider registriert werden (Instanz einer Unterklasse von __Provider und Instanz der Klasse __EventConsumerProviderRegistration: __ProviderRegistration, in die dann die unterstützten (logischen) __EventConsumer eingetragen werden (Attribut:ConsumerClassNames)).
- Ein __EventFilter muß definiert werden.
- Die erwünschten Verknüpfungen müssen als Instanzen der Assoziations Klasse *__FilterToConsumerBinding* eingetragen werden.

5.3.2 Implementierung

Beschreibung der Anforderungen

Es soll der prinzipielle Mechanismus der Event-Behandlung gezeigt werden. Dabei sollen im laufenden Betrieb Events ausgelöst und anschließend sichtbar gemacht werden. Dies schließt sowohl Windows NT interne bzw. CIMOM interne Events wie auch DMI Events mit ein.

Beschreibung der Lösung

Da hier Hauptaugenmerk auf die Funktion des Mechanismus gelegt wurde, wurden die mitgelieferten Provider verwendet. Dies sind zum einen der RegistryEventProvider sowie der CIMOM (für interne Events) und der DMIEventProvider, die Notifications senden. Zur Anzeige dient die Beispiel-Applikation MS WBEM Event Viewer. Hier wurde Consumer-Provider und Sink in einer EXE-Datei zusammengefaßt (eventviewer.exe). Bei der Realisierung wurde eine MOF-Datei (Event.mof) erzeugt, die sämtliche Definitionen der benötigten Klassen und Instanzen enthält. Vor dem Compilieren (im CIMOM registrieren), müssen die mitgelieferten Dateien regevent.mof und dmievent.mof compiliert werden, da damit die Registry-Events und die DMIEvents sowie der RegistryEventProvider und der DMIEventProvider registriert werden.

Zunächst müssen die logischen Event-Consumer registriert werden. Dazu wird zunächst eine Klasse *EventViewerConsumer* von der Klasse __EventConsumer abgeleitet. Dabei werden die Attribute *Name, Severity* und *Description* eingeführt. Das Attribut *Name* ist dabei das Schlüsselattribut der Klasse.

```
//////////////////////////////////////////////////////////////////
/// Class: EventViewerConsumer
/// Derived from: __EventConsumer
//////////////////////////////////////////////////////////////////
class EventViewerConsumer : __EventConsumer
{
      [key] string Name = "";
      [read] uint8 Severity = 0;
      [read] string
      Description = "";
};

//////////////////////////////////////////////////////////////////
/// Registration of an EventConsumer
/// This is a logical Consumer which is bound to a Consumer-Provider
/// in __SystemClass.__ProviderRegistration.
/// __EventConsumerProviderRegistration
//////////////////////////////////////////////////////////////////
instance of EventViewerConsumer
{
     Name = "FopraConsumer1";
     Description = "Änderungen unter
                    \\\\HKEY_LOCAL_MACHINE\\SOFTWARE\\Fopra1";
};
```

Mit dem Schlüsselwort *instance of* wird eine Instanz der Klasse *EventViewerConsumer*, die oben neu definiert wurde, erzeugt. Dies dient dazu, den logischen Event Consumer, den diese Instanz darstellt, im CIMOM zu registrieren. Als kennzeichnenden Namen wurde im Beispiel hier „FopraConsumer1“ verwendet. Die eingetragene Beschreibung (Description) dient später der Anzeige im Event Viewer, damit zu sehen ist, welcher Event ausgelöst wurde.

Das selbe geschieht für FopraConsumer2 (hier: Key=…\Fopra2), FopraConsumer3 (hier: Description = „Änderungen im CIMOM Repository“) und FopraConsumer4 (hier: Description = „DMI Event“).

Anschließend muß die Klasse der Consumer an einen Provider gebunden werden:

```
//////////////////////////////////////////////////////////////////
/// Instances of: __EventConsumerProviderRegistration
//////////////////////////////////////////////////////////////////
instance of __EventConsumerProviderRegistration
{
      ConsumerClassNames = {"EventViewerConsumer"};
      provider="\\\\.\\root\\cimv2:__Win32Provider.Name=
\"EventViewerConsumer\"";
};
```

Mit dem Schlüsselwort *instance of* wird eine Instanz der Klasse *EventConsumerProviderRegistration* erzeugt. Dies dient dazu, die logischen Event Consumer, die von der Klasse *EventViewerConsumer* vertreten werden, an einen Provider zu binden. Dazu müssen der Klassenname und der Provider angegeben werden (ConsumerClassNames, provider).

Nun können die Filter definiert werden:

```
//////////////////////////////////////////////////////////////////
// Registration of an Event-Filter
/// Here: All Changes of the Registry Tree under
/// \\HKEY_LOCAL_MACHINE\Software\Fopra1
//////////////////////////////////////////////////////////////////
instance of __EventFilter
{
      Name = "FopraEventFilter1";
      Query = "select * from RegistryTreeChangeEvent where
                Hive = \"HKEY_LOCAL_MACHINE\" and
                RootPath = \"Software\\\\Fopra1\"";
      QueryLanguage = "WQL";
};
```

Event Filter sind Instanzen der Klasse __EventFilter. Das kennzeichnende Attribut ist hier der *Name*. Mit dem Attribut *QueryLanguage* wird die Möglichkeit offengehalten, verschiedene Abfragesprachen zu verwenden. Das MS WBEM SDK unterstützt derzeit jedoch lediglich WQL. Das wichtigste Attribut ist *Query*. Hier wird definiert, welche Events betrachtet werden sollen. Im Beispiel hier werden alle Events betrachtet (select *), die der Klasse *RegistryTreeChangeEvent* angehören (from RegistryTreeChangeEvent), d.h. alle Änderungen der Registry, die folgenden Bedingungen genügen (where):
Hive = HKEY_LOCAL_MACHINE, d.h. es wird nur der Teilbaum unter HKEY_LOCAL_MACHINE betrachtet,
RootPath = Software\\Fopra1, d.h. unter HKEY_LOCAL_MACHINE befindet sich der Schlüssel SOFTWARE\\Fopra1 und nur Änderungen, die diesen Teilbaum betreffen werden betrachtet.

Der FopraEventFilter2 sieht entsprechend aus und wird daher hier nicht extra aufgeführt.

```
//////////////////////////////////////////////////////////////////
// Registration of a third Event-Filter
/// Here: All Changes of of class definitions in the CIMOM Repository
//////////////////////////////////////////////////////////////////
instance of __EventFilter
{
      Name = "FopraEventFilter3";
      Query = "select * from __ClassOperationEvent";
      QueryLanguage = "WQL";
};
```

Der dritte EventFilter beschränkt sich auf alle Klassenoperationen im CIMOM-Repository, z.B. anlegen, löschen, umbenennen von Klassen.

```
//////////////////////////////////////////////////////////////////
// Registration of a fourth Event-Filter
/// Here: All DMI Events
//////////////////////////////////////////////////////////////////
instance of __EventFilter
      Name = "FopraEventFilter4";
      Query = "select * from DmiEvent";
      QueryLanguage = "WQL";
};
```

Der vierte EventFilter betrachtet alle auftretenden DMI-Events.

Schließlich müssen die Consumer noch mit den Filtern verknüpft werden:

```
//////////////////////////////////////////////////////////////////
/// Registration of the Filter-Consumer-Binding for Consumer1
/// Instances of this Class combine a Consumer with a Filter
/// Here: FopraConsumer with FopraEventFilter
//////////////////////////////////////////////////////////////////
instance of __FilterToConsumerBinding
{
      Consumer = "EventViewerConsumer.Name=\"FopraConsumer1\"";
      DeliverSynchronously = FALSE;
      Filter = "__EventFilter.Name=\"FopraEventFilter1\"";
};
```

Da es sich um eine Assoziations Klasse handelt (siehe Kapitel 6), werden hier Referenzen auf je einen Consumer und einen Filter angegeben. Das Attribut *DeliverSynchronously* gibt an, ob die Notification synchron oder asynchron übertragen werden soll. In der Regel wird man hier asynchrone Zustellung wählen, so auch im Beispiel.
Der Vorteil der synchronen Übertragung ist, daß das Senden der Notification vom Provider zum CIMOM und von diesem zum Consumer im gleichen Thread geschieht und daher kein Kontext-Wechsel nötig ist. Allerdings wird der Provider für die Zeit bis zur vollendeten Zustellung blockiert, weshalb die Empfehlung gegeben wird, synchrone Zustellung nur bei sehr schnellen Consumern (deren Bearbeitungszeit von Notifications kürzer als 20 Millisekunden beträgt) in Erwägung zu ziehen.

Da die Instanzen für FopraConsumer2, FopraConsumer3 und FopraConsumer4 entsprechend aussehen, wurden sie hier ebenfalls weggelassen.
Die Realisierung verwendet demzufolge einen Sink für alle drei Consumer.
Die unterschiedlichen Events werden alle in einem gemeinsamen Fenster dargestellt.

Zu Demonstrationszwecken wurden im Folgenden einige Events ausgelöst:

- Der Key *Test* wurde in der Registry unter *\\HKEY_LOCAL_MACHINE\Software\Fopra1* angelegt. Die Events wurden durch das Anlegen sowie das Umbenennen (von *NewKey #1* in *Test*) ausgelöst.
- Der Value *NewValue #1* wurde unter *\\HKEY_LOCAL_MACHINE\Software\Fopra2* angelegt.
- Die Klasse *TestClass* wurde im CIMOM als Unterklasse von *Fopra* angelegt.
- Der Klasse *TestClass* wurde das Attribut Name hinzugefügt.
- Der oben angelegte Registry-Key und -Value wurden gelöscht.

Das Ergebnis ist in folgendem Fenster zu sehen:

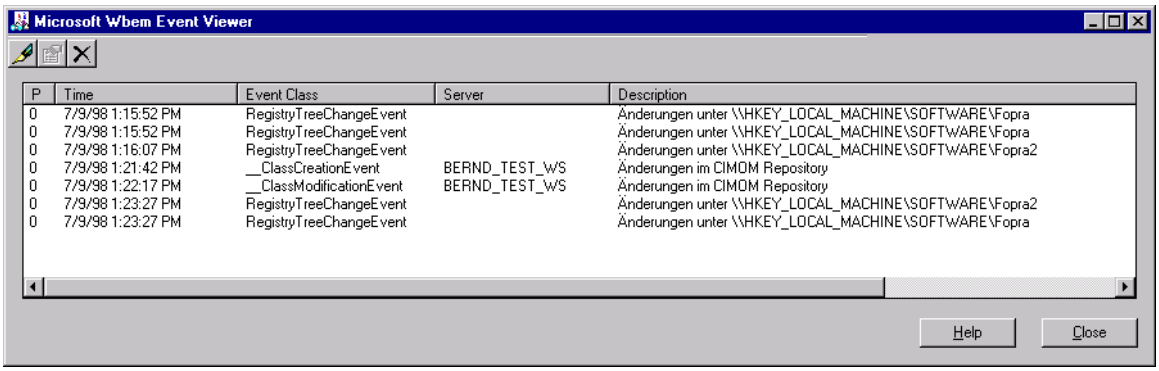

Abbildung 5-6: Verschiedene Events

Probleme und Hindernisse bei der Implementierung

- Ist eine Filterdefinition (Query) fehlerhaft, so akzeptiert CIMOM diese trotzdem. Allerdings läßt sich dieser Filter dann nicht mit einem Consumer verknüpfen (__FilterToConsumerBinding).
- CIMOM liefert keine genauen Informationen, warum er bestimmte Query-Definitionen nicht zuläßt. Etwas genauer sind die Fehlermeldungen beim Wbem Event Viewer (s.o.). Versucht man hier dieselben Verknüpfungen von Hand anzulegen, bekommt man unter Umständen Meldungen wie *„Query too broad“* oder *„Query too specific“*. Dies bedeutet jedoch nicht, daß die Ursache für den Fehler daraus unbedingt erkennbar wäre.
- Auch ist undokumentiert, daß trotz Referenz auf eine Klasse (s. Assoziations Klassen) nicht unbedingt ein Attribut aus dieser Klasse zur Identifikation von Instanzen angegeben werden muß. Es muß dagegen das Attribut mit der Kennzeichnung *Key* verwendet werden, entsprechend mit der Klassenbezeichnung, in der es definiert ist. Dies kann sowohl die referenzierte, wie auch eine Vater- oder Sohn-Klasse sein. Dafür ist es dann auch notwendig, bei der Referenz den Klassennamen und das Attribut mit anzugeben, etwa Consumer="EventViewerConsumer.Name=\"FopraConsumer2\"";

5.4 Win32 Provider

Mit dem MS WBEM SDK werden eine Reihe von NT spezifischen Providern mitgeliefert. Sie sind alle Standard Provider und werden alle als Win32Provider registriert.
Folgende Provider werden mit Build 486 ausgeliefert:

- RegistryEventProvider: *s. 5.3 Events*; dieser Provider liefert Notifications bei Zugriff auf die Registry. Hier handelt es sich um einen *EventProvider*.
- EventViewerConsumer: *s. 5.3 Events*; dieser Provider stellt zwei DCOM-Objekte zum Empfang und zur Darstellung von Event-Notifications zur Verfügung. Es handelt sich dabei um einen *EventConsumerProvider*.
- RegProv: der Registry Provider erfaßt Registry-Keys (*InstanceProvider*).
- CIMWin32: Der Standard-Provider liefert Informationen über die laufende Windows-Installation: Prozesse, Services, Printjobs, etc. (*InstanceProvider, MethodProvider (er ist zwar als solcher registriert, jedoch wurden Methoden bisher (Build 486) noch nicht implementiert)).*
- MS_NT_EVENTLOG_PROVIDER: Dieser Provider liefert Einträge des EventLog als Instanzen und kann das EventLog sichern und leeren (*InstanceProvider, MethodProvider).*
- MS_NT_EVENTLOG_EVENT_PROVIDER: Dieser *EventProvider* liefert Notifications für NT-Events (die auch ins EventLog geschrieben werden).
- PerfProv: Der Performance Monitor Provider wertet die Daten des Windows NT Performance Monitors aus.
- DMI Provider: Der DMI Provider liefert DMI-konforme Informationen an den CIMOM.
- SNMP-Provider: Der SNMP-Provider stellt die Schnittstelle zur SNMP-Welt dar.

Auf *SNMP-Provider, DMI-Provider, RegistryEventProvider* und *EventViewerConsumer* wurde in den vorangegangenen Kapiteln eingegangen. Der *MS_NT_EVENTLOG_PROVIDER* entspricht von der Funktionsweise dem *Registry Provider (RegProv)*, der *MS_NT_EVENTLOG_EVENT_PROVIDER* dem *RegistryEventProvider*. Daher wird hier nicht näher auf diese beiden Provider eingegangen. Die Auswahl ist nicht rein zufällig, sondern historisch bedingt: Die beiden EventLog Provider waren in der Build 220 noch nicht vorhanden.
Der *Performance Monitor Provider* funktioniert in der selben Art und Weise wie der Registry Provider.

Die Provider CIMWin32, RegProv, PerfProv und RegistryEventProvider sind alle Bestandteil der stdprov.dll, der EventViewerConsumer ist in der eventviewer.exe realisiert und MS_NT_EVENTLOG_EVENT_PROVIDER und MS_NT_EVENTLOG_PROVIDER befinden sich in ntevt.dll.

5.4.1 Registry Provider

Beschreibung

Der Registry Provider kann als *Instance Provider* oder als *Property Provider* arbeiten. Er liefert also entweder zu gegebenen Klassen dynamisch die Instanzen, oder liest und belegt die Attribute von statischen Instanzen.
Der Provider besteht aus Funktionen der stdprov.dll, die mittels API-Aufrufen auf die Registry zugreifen. Dabei kann es sich um eine lokale wie um eine Remote Registry handeln.

Registrierung

Der Registry Provider muß an zwei Stellen registriert werden:
- Unter __Provider
- Unter __ProviderRegistration

Er wird dabei als Instanz der folgende Klassen im CIMOM registriert:
- __Win32Provider: Hier sind der Name sowie die Identifikation in der Registry anzugeben.
- __InstanceProviderRegistration: Hier werden die Fähigkeiten des Providers angegeben (z.B. ob er lesend oder schreibend auf Instanzen zugreifen kann).

Implementierung

Beschreibung der Anforderungen

Es sollen Registry Keys ausgelesen und verändert werden um die Funktion des Registry Provider als Instance Provider zu demonstrieren.

Beschreibung der Lösung

Es werden hier die in der Registry verzeichneten Services ausgelesen. Dies hat den Hintergrund, daß diese zusammen mit den noch folgenden Prozessen (CIMWin32) die Grundlage für die beispielhafte Implementierung einer Assoziations Klasse (Kapitel 6) bilden.

Zur Darstellung dient hier das *WBEM Developer Studio*. Hier können die Instanzen dynamisch aufgerufen und angezeigt werden.

Da diese drei Beispiele in Kapitel 6 zusammengefaßt werden, bietet sich an, der Übersicht halber eine Container-Klasse zu definieren, in die dann alle weiteren Klassendefinitionen eingehängt werden:

```
//////////////////////////////////////////////////////////////////
// Declare a class called FOPRA as Container for all new classes
//////////////////////////////////////////////////////////////////
[Abstract]
class Fopra
{
};
```

Der Qualifier *Abstract* gibt an, daß diese Klasse keine eigenen Instanzen enthält.

Zunächst muß der Registry Provider registriert werden, da er nicht Bestandteil der Standarddefinition (cimwin32.mof) ist:

```
//////////////////////////////////////////////////////////////////
// Declare an instance of the __Win32Provider so as to "register" the
// Registry Provider.
//////////////////////////////////////////////////////////////////
instance of __Win32Provider
{
    ClientLoadableClsid = NULL;
    Name        = "RegProv" ;
    ClsId         = "{fe9af5c0-d3b6-11ce-a5b6-00aa00680c3f}" ;
} ;

instance of __InstanceProviderRegistration
{
      Provider    =
"\\\\.\\root\\CIMV2:__Win32Provider.Name=\"RegProv\"";
      SupportsEnumeration     = true;
      SupportsGet = true;
      QuerySupportLevels      = {"WQL:UnarySelect"};
};
```

Das Attribut *ClsId* enthält die Kennzeichnung des Providers, mit der er in der Registry registriert ist.
Der Registry Provider unterstützt sowohl die Aufzählung (SupportsEnumeration) als auch das Lesen der Attribute (SupportsGet) von Instanzen (hier: der Klasse *RegServices*).
Zu beachten ist hier, daß das Attribut *SupportsGet* fehlt, da der mitgelieferte Provider nicht in der Lage ist, in die Registry zu schreiben.

Schließlich wird noch eine Klasse *RegServices* definiert, deren Instanzen dann mit Registry Keys gefüllt werden:

```
//////////////////////////////////////////////////////////////////
// Declare a class to get the Services from the Registry
//////////////////////////////////////////////////////////////////
[dynamic, provider("RegProv"),ClassContext
("local|hkey_local_machine\\system\\CurrentControlSet\\Services")]
class RegServices as $RegistratedService : Fopra
{
      [PropertyContext("TransportsDeviceName"), key] String Name = "";
      [PropertyContext("DisplayName")] String DisplayName;
      [PropertyContext("Start")] String Start;
};
```

Die Klasse wird dynamisch von dem Provider RegProv, also dem Registry Provider gefüllt (Qualifier *dynamic, provider*).
Hier ist anzumerken, daß der Registry Provider stets einen ClassContext benötigt. Hier wird angegeben von welchem Key die Subkeys aufgezählt werden sollen. Im Falle der Benützung als Property Provider wird hier der zu bearbeitende Key angegeben. Hier ist der ClassContext *local|HKEY_LOCAL_MACHINE\\System\\CurrentControlSet\\Services,* also genau die in der Registry (des lokalen Rechners) verzeichneten Services.
Die vom Provider ausgelesenen Werte (PropertyContext(„...“)) werden auf Attributsnamen abgebildet (Name, DisplayName und Start).

In der folgenden Abbildung ist die Aufzählung der Subkeys (als Instanzen der Klasse RegServices) im WBEM Developer Studio zu sehen.

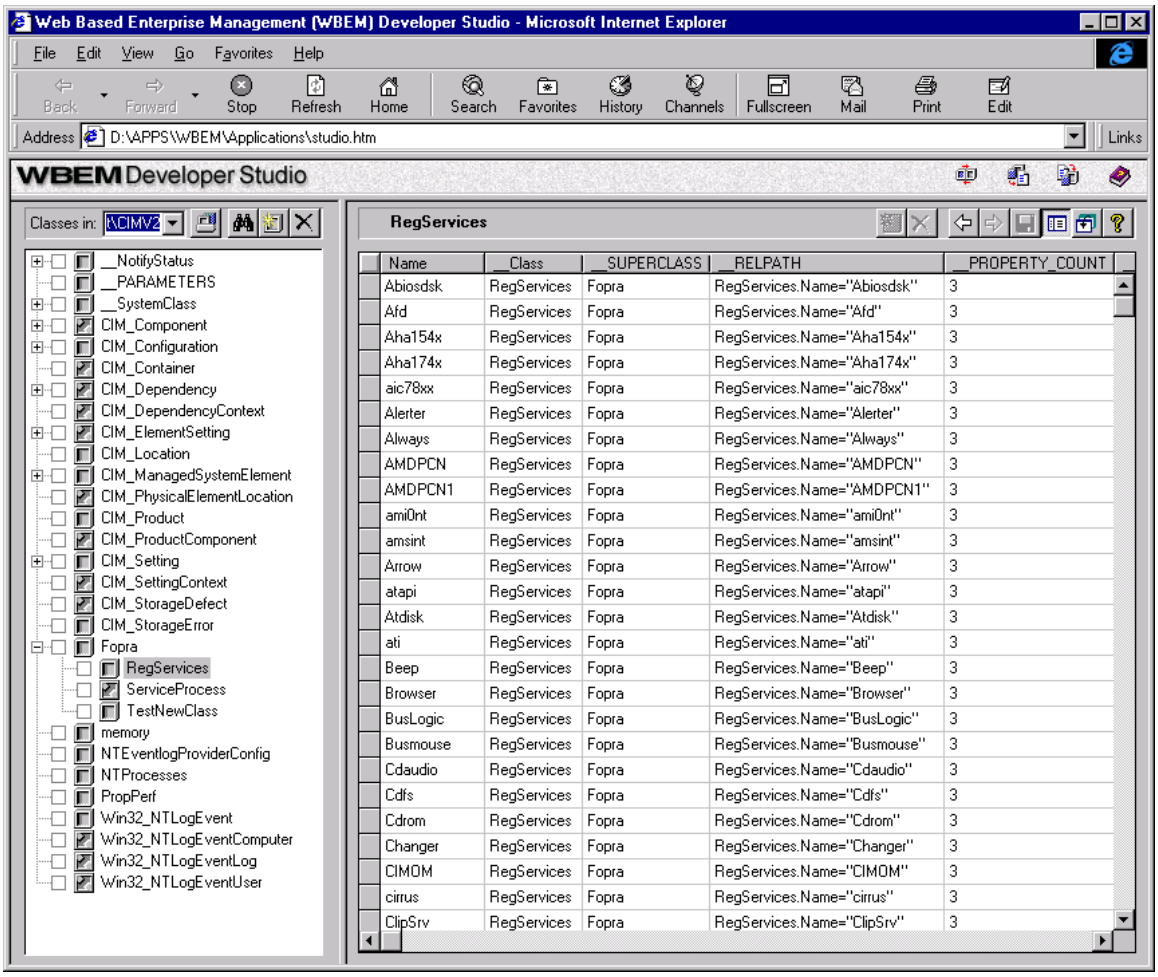

Abbildung 5-7: In der Registry verzeichnete Services

Probleme und Hindernisse bei der Implementierung

- Die Dokumentation des Build 486 ist noch auf dem Stand von Build 220 (betrifft ebenso PerfProv). In der Zwischenzeit hat sich allerdings die Provider-Registrierung erheblich geändert. Das hat zur Folge, daß die spärlich vorhanden Beispiele nicht lauffähig sind.
- So Beispiele vorhanden sind, sind sie voll von Syntaxfehlern oder nicht mehr gültigen Attributen.
- Der mitgelieferte Provider RegProv ist nicht in der Lage, Werte zu setzen (PutInstance = false).

5.4.2 CIMWin32

Beschreibung der Architektur

Der CIMWin32 kann als Instance Provider oder als Property Provider arbeiten. Er liefert also entweder zu gegebenen Klassen dynamisch die Instanzen, oder liest und belegt die Attribute von statischen Instanzen.
Der Provider besteht aus Funktionen der stdprov.dll, die auf die Registry zugreifen und mit Win32-API-Aufrufen die benötigten Informationen sammeln.
Die Informationen erstrecken sich auf eine Vielzahl der vom Betriebssystem bereitgestellten Daten wie Filesystem, Directories, PrintJobs, Services, Prozesse, Threads, Betriebssystem, Accounts, Systemeinstellungen, Adapter, Festplatten oder Chipsatz.

Registrierung

Der CIMWin32 muß nicht eigens registriert werden, dies geschieht bereits bei der Installation. Sollte es aus irgendeinem Grund doch nötig sein, so findet sich die Definition in cimwin32.mof.
Im Übrigen entspricht die Registrierung der der anderen Provider.

Implementierung

Beschreibung der Anforderungen

Es sollen Informationen über das laufende System ausgelesen werden.

Beschreibung der Lösung

Im Hinblick auf Kapitel 6 werden hier die laufenden Prozesse ausgelesen. Die Definition befindet sich ebenfalls in cimwin32.mof.

Zur Darstellung dient hier das *WBEM Developer Studio*. Hier können die Instanzen dynamisch aufgerufen und angezeigt werden.

Verwendet wird die Klasse
CIM_ManagedSystemElement\CIM_LogicalElement\CIM_Process\Win32_Process:

In der folgenden Abbildung ist die Auflistung der gerade laufenden Prozesse zu sehen.

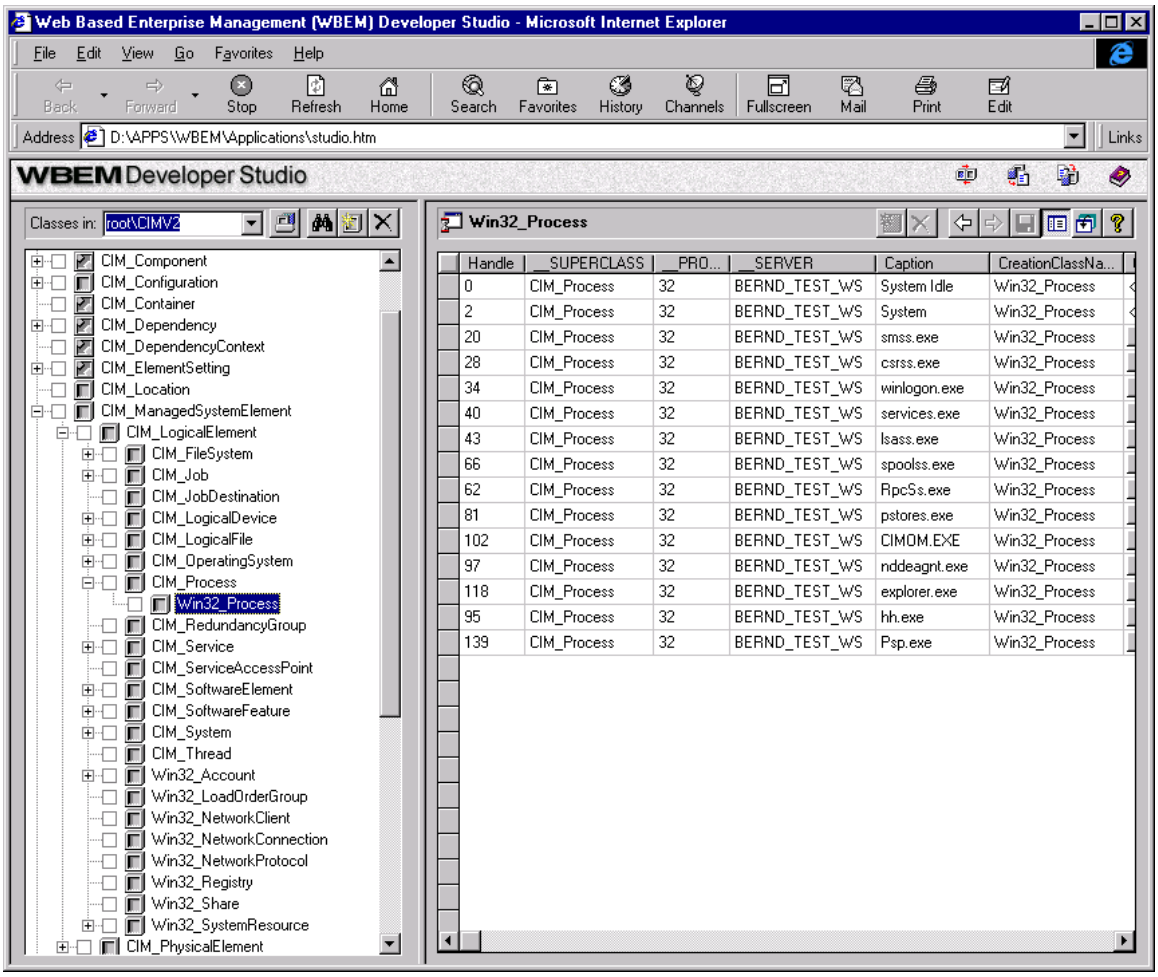

Abbildung 5-8: Gerade laufende Prozesse

Probleme und Hindernisse

Unterschiedliche Handhabung in Build 220 und Build 486:

- In Build 220 wurden eine Reihe verschiedener Klassen mit Prozessen definiert. Jedoch wurde nur eine Klasse von dem Standard Provider mit Instanzen gefüllt.
- In Build 486 wurden diese Klassen zu einer zusammengefaßt.
- Durch die komplexen Strukturen (3-stufige Vererbung, zahlreiche Verknüpfungen) ist es jedoch kaum möglich, die Klasse aus dem Schema herauszulösen, da sonst der CIMWin32 nicht mehr in der Lage ist, die Klasse *Win32_Process* mit Instanzen zu füllen. Er benötigt scheinbar nicht nur alle Attribute (auch die vererbten), sondern auch sämtliche mit Win32_Process verknüpfte Klassen (und deren Verknüpfungen und so fort…). Da dies dann aber das gesamte Win32 Schema abdeckt, wurde hier die Orginalklasse verwendet, da dann eine Separierung sinnlos wird.

6 Integration von Informationen

Um Informationen verschiedener Provider miteinander zu verknüpfen gibt es prinzipiell mehrere Möglichkeiten.
Eine Möglichkeit ist, zu gegebenen Klassen die Instanzen statisch zu erzeugen, d.h. die Instanzen sind dann fest im Repository gespeichert. Dann können die Attribute dieser Instanzen von verschiedenen *Property Providern* gesetzt werden.
Es können jedoch auch sogenannte *Assoziations Klassen* verwendet werden. Diese Klassen sind Referenzen auf Klassen, deren Instanzen dynamisch erzeugt werden können.
Im folgenden werden wir uns auf die Assoziations Klassen beschränken, da sie die weitaus universellere Lösung darstellen.

6.1 Assoziations Klassen

Ein mächtiges und wichtiges Instrument der WBEM-Architektur stellen die Assoziations Klassen dar.
Sie ermöglichen die Abbildung von n:m-Verknüpfungen. Somit können auch komplexe Zusammenhänge beschrieben werden.
Ein anderes wichtiges Einsatzgebiet von Assoziationsklassen ist die Verknüpfung von Informationen, die aus verschiedenen Quellen stammen.

Beschreibung

Die Attribute von Assoziations Klassen enthalten Referenzen auf andere Klassen. Instanzen dieser Assoziations Klassen zeigen damit auf Instanzen anderer Klassen. Durch sie ist also eine Möglichkeit gegeben, „durch das System zu navigieren“.
Ausgezeichnet werden sie durch den *Qualifier* „Association“.
Die Instanzen können entweder statisch angelegt werden oder durch einen entsprechenden *Instance Provider* gefüllt werden. Dieser muß dann aber in der Lage sein, entsprechend sinnvolle Referenzen zu liefern.

Implementierung

Beschreibung der Anforderungen

Es sollen Informationen aus verschiedenen Quellen miteinander kombiniert und dargestellt werden. Dabei soll die prinzipielle Funktionsweise anhand eines Beispiels herausgestellt werden.

Beschreibung der Lösung

Da die Dokumentation im Bezug auf die Realisierung von Providern äußerst dürftig und zudem auch nicht auf dem aktuellen Stand (Build 486) ist, wurde darauf verzichtet, einen entsprechenden Provider zu schreiben.
Statt dessen wurde stellvertretend eine statische Instanz erzeugt, an der sich die Funktionsweise demonstrieren läßt.

Zur Darstellung wurde das im MS WBEM SDK mitgelieferte CIM Studio eingesetzt.
Hier können auch die Verknüpfungen graphisch ausgegeben werden und anhand der graphischen Ausgabe kann auch zwischen den Klassen bzw. den Instanzen navigiert werden.

Zuerst wurde die Klasse *ServiceProcess* definiert.
Zweck dieser Klasse ist, eine Zuordnung von laufenden Prozessen zu registrierten Services zu ermöglichen.
Sie ist unter der schon bekannten Container-Klasse Fopra eingehängt.

Die Referenzen zeigen dabei auf die in Kapitel 5.4 eingeführte Klasse *RegServices* (die in der Registry verzeichneten Services), die Klasse *Win32_Service* (wird von dem Provider CIMWin32 gefüllt, enthält die laufenden Services) und die Klasse *Win32_Process* (siehe Kapitel 5.4, laufende Prozesse).

```
////////////////////////////////////////////////////////////////////
// Declare an Association Class to gather the information
// Win32_Service and Win32_Process are built-in classes
////////////////////////////////////////////////////////////////////
[Association]
class ServiceProcess:Fopra
{
      [Multiplicity ("MV") , Key, Volatile]
      Win32_Service REF Service;
      [Multiplicity ("MV") , Key, Volatile]
      Win32_Process REF Process;
      [Multiplicity ("MV") , Key, Volatile]
      RegServices REF RegistratedAs;
};
```

Das Schlüsselwort *REF* weist dabei einem Attribut (z.B. RegistratedAs) eine Referenz auf eine andere Klasse zu (z.B. RegServices).
Wie bereits erwähnt, wurde dann eine statische Instanz erzeugt. Da davon auszugehen ist, daß auf allen Testrechnern der CIMOM-Service läuft, wurde dieser für das Beispiel verwendet.
Die Instanz der Klasse ServiceProcess zeigt damit auf die Instanzen des registrierten sowie des laufenden Services „CIMOM“ sowie auf den laufenden Prozess dieses Services, „CIMOM.EXE“.

```
////////////////////////////////////////////////////////////////////
// Instead of writing a provider, static instances are declared to show
// prototypically the mechanism
////////////////////////////////////////////////////////////////////
instance of ServiceProcess
{
      Process = "Win32_Process.Name=\"CIMOM.EXE\"";
      Service = "Win32_Service.Name=\"CIMOM\"";
      RegistratedAs = "RegServices.Name=\"CIMOM\"";
};
```

Zur Darstellung wird das WBEM Developer Studio gestartet.
Zunächst ist die Klasse ServiceProcess und deren (einzige) Instanz auszuwählen.
Danach muß die Karteikarte *Associations* aktiviert werden.

Dann ist die graphische Darstellung der Verknüpfung zu sehen. Zu beachten ist hier, daß die referenzierten Instanzen der Partner-Klassen bereits explizit angezeigt werden.

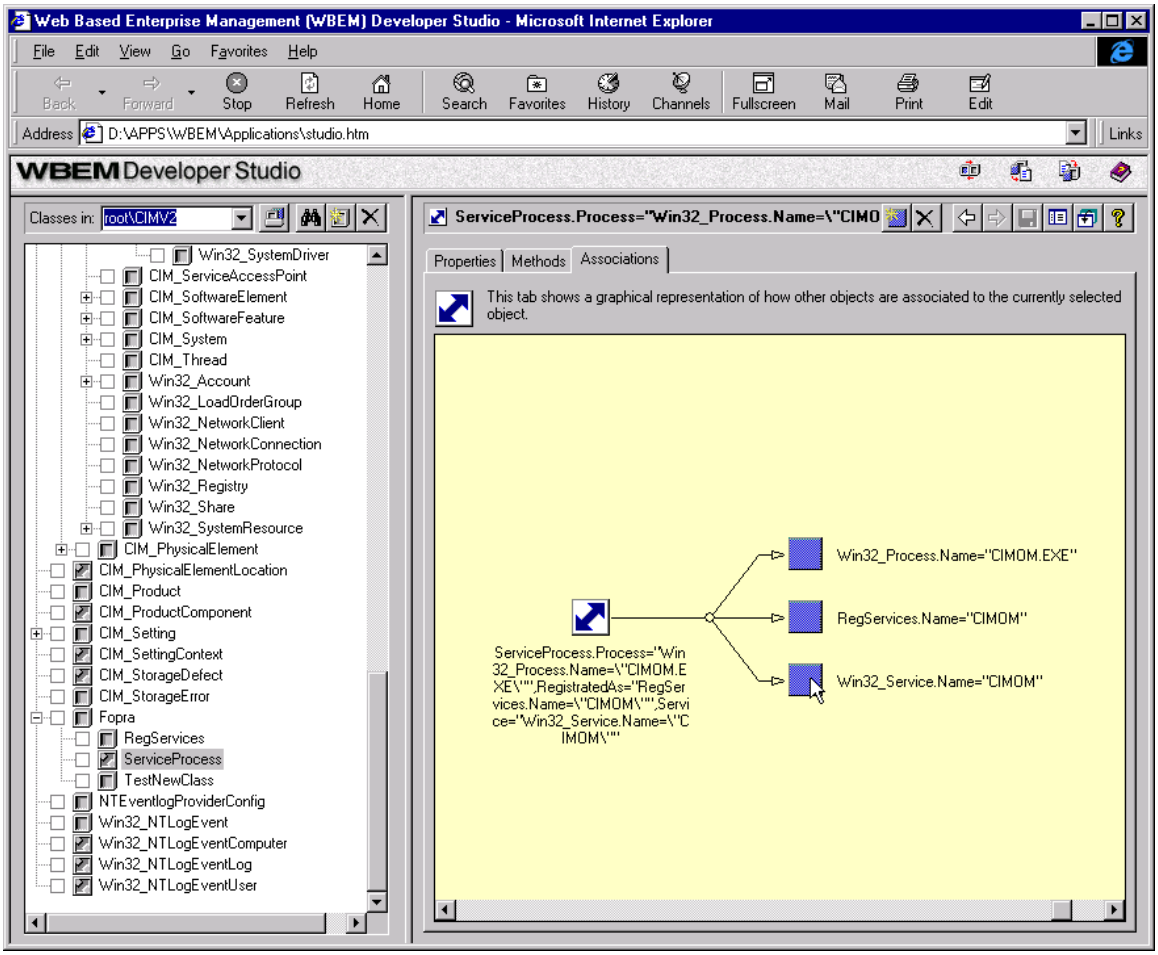

Abbildung 6-1: Assoziations Klasse *ServiceProcess*

In der graphischen Darstellung wird dann der Win32_Service „CIMOM“ ausgewählt. Daraufhin wechselt die Darstellung auf die Verknüpfungen dieser Instanz (Die Kästchen mit den Pfeilen stellen Assoziations Klassen dar, die ausgefüllten Kästchen normale Klassen).

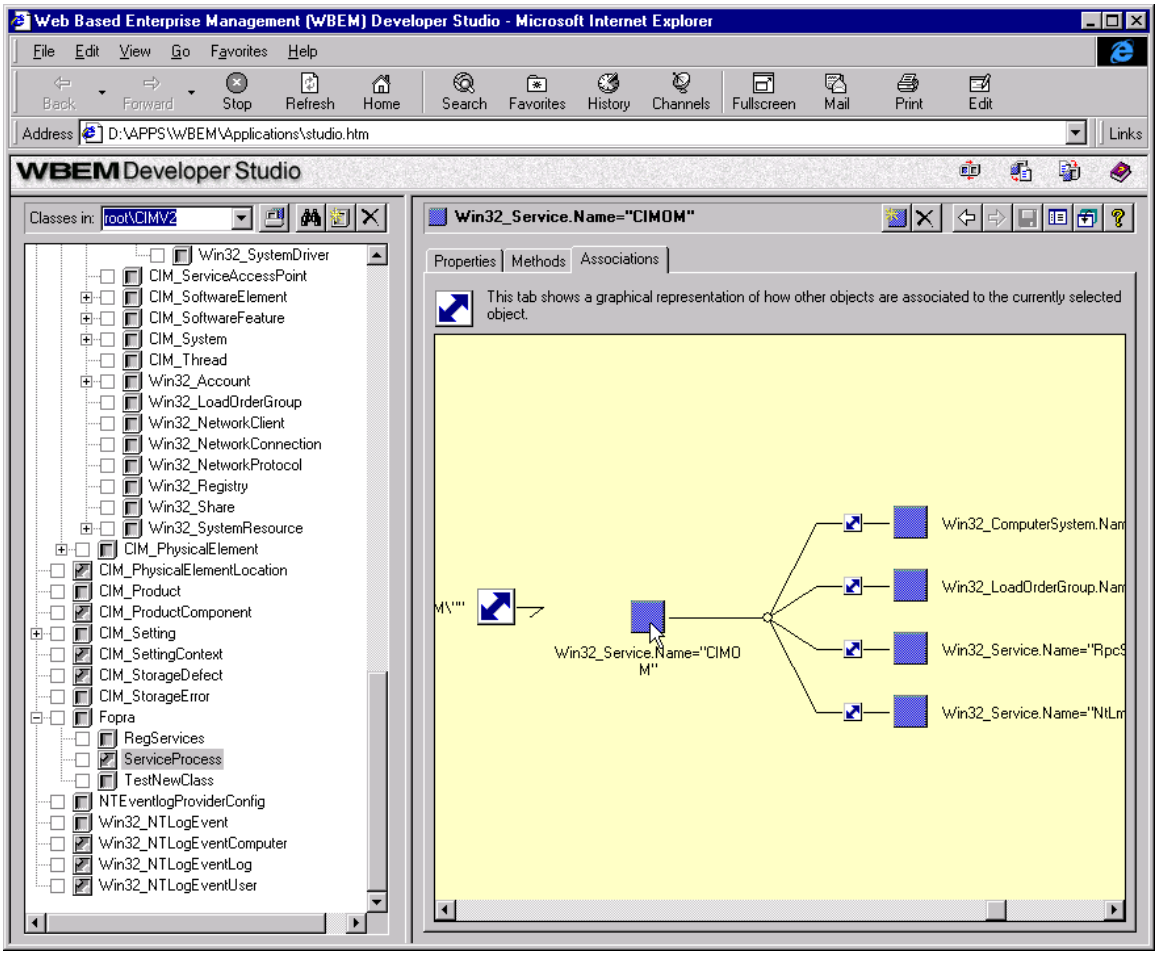

Abbildung 6-2: Anwählen des Services

Wählt man nun die Karteikarte *Properties,* so erhält man die Attribute (und deren Werte) der verknüpften Instanz, hier also des laufenden Services „CIMOM“.

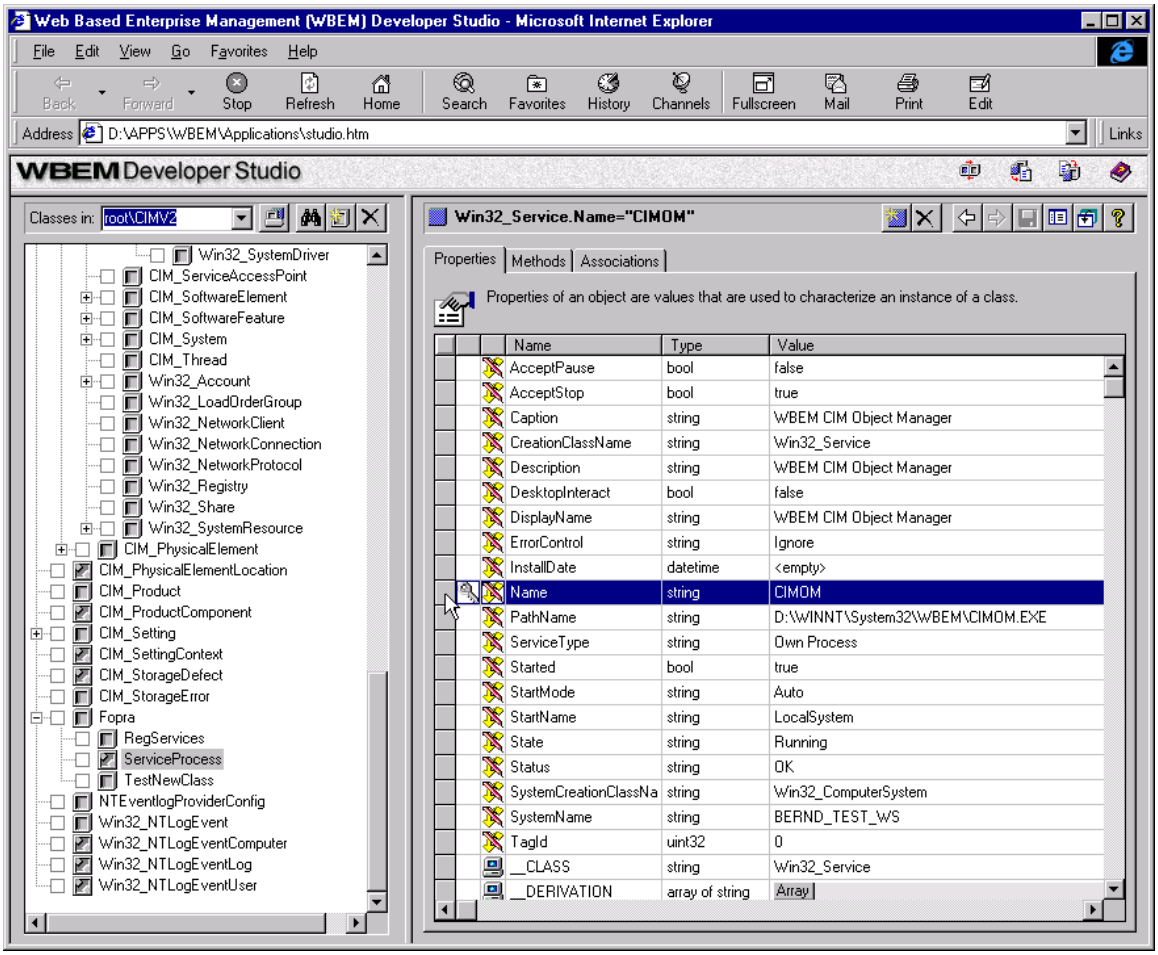

Abbildung 6-3: Attribute des ausgewählten Service

Probleme und Hindernisse bei der Implementierung

Hauptproblem war auch hier die sehr schlechte Dokumentation. Die als Beispiel aufgeführten (und mitgelieferten) Klassen lassen sich teilweise nicht benützen, da sie entweder fehlerhaft definiert sind oder aber der entsprechende Provider nicht die gewünschte Funktionalität erbringt.
Das führte auch zu dem Entschluß, keinen Provider zu schreiben, da selbst die mitgelieferten Provider nicht voll funktionstüchtig sind. Zudem liegen sie nicht als Quellcode vor und sind nicht dokumentiert.
Build 486 brachte zwar insofern eine Verbesserung, daß die Assoziations Klassen prinzipiell funktionieren (bei Build 220 war auch das nicht gewährleistet).

7 Fazit

WBEM ist ein interessanter Ansatz, um Management heterogener Umgebungen zu realisieren. Es wird vor allem Wert darauf gelegt, bestehende Lösungen zu integrieren, was auf dem Gebiet des Netz- und Systemmanagements ein nicht zu unterschätzender Vorteil ist.
Da die vorliegenden Versionen noch Beta-Stadium haben, kann man nicht erwarten, daß alle Funktionen schon fehlerfrei implementiert sind. Besonders die doch sehr dürftige Dokumentation, die durchwegs auch die Hauptprobleme dieser Arbeit bereitete, muß in der Final Version noch einen großen Schritt machen. Laut Ankündigungen der Firma Microsoft soll die Final Version noch im Juli 1998 veröffentlicht werden. Abzuwarten bleibt, inwieweit sie die Erwartungen erfüllen kann.
Im Gegensatz zu anderen Architekturen hat WBEM jedoch den Vorteil, daß es bald verfügbar sein wird. Damit steht eine erste Implementierung, mit der Erfahrungen gesammelt werden können zur Verfügung.
Die Entwicklung des Netz- & Systemmanagements hat gezeigt, daß sich meist Lösungen durchsetzen, für die es früh Implementierungen gibt.
Die Integration von WBEM in die Microsoft Betriebssysteme Windows 98 und Windows NT 5.0 wird es schnell zu einer weiten Verbreitung von WBEM als Managementplattform beitragen.
Ein Hauptmakel ist das fehlen einer offenen Schnittstelle wie sie mit der Java API in der ersten Beta Version angekündigt worden ist. In er vorliegenden Beta Version ist die Java-Schnittstelle stillschweigend aus dem MS WBEM SDK verbannt worden. Damit geht ein Hauptgesichtspunkt des gesamten Ansatzes verloren.
Es ist nur noch mit größtem Aufwand möglich, das Management von einem beliebigen Webbrowser auf einer unabhängigen Plattform zu steuern. Damit geht in unseren Augen auch der Gedanke des Web Based Managements verloren.
Dies hat sicherlich strategische Gründe, dennoch stellt sich die Frage, ob nicht damit einem vielversprechenden Ansatz der Zugang zu anderen Umgebungen (wie v.a. Unix) von vornherein verwehrt wird. Allerdings haben schon andere namhaft Firmen wie HP und IBM angekündigt, WBEM zu unterstützen. Vielleicht kann ja der „Web-Gedanke“ dadurch wieder zum Leben erweckt werden.
Insgesamt läßt sich feststellen, daß ein guter Ansatz, zumindest teilweise, über die bloße Definition hinausgeht und mit dem MS WBEM SDK eine erst Implementierung existiert.
Die Akzeptanz muß sich dieser Ansatz nun erlangen.

Anhang

Installation des MS WBEM SDK

Die Installation des SDKs ist vor allem in der vorliegenden Beta-Version etwas komplex und teilweise ist sogar das Setup-Programm mit Fehlern behaftet. Es bleibt zu erwarten, daß in der Release-Version ein verbessertes Setup mitgeliefert wird.
Folgende Systemvoraussetzungen werden von Microsoft empfohlen:

- Intel-basierte Workstation oder Server welche(r) die Mindestvoraussetzung für die Installation von Windows NT 4.0 erfüllt
- mindestens 32MB Speicher
- ca. 30MB Speicherplatz auf der Festplatte
- für Testrechner wird mindestens ein Pentium 133 Prozessor empfohlen für Entwicklungsrechner ein Pentium 166 Prozessor

Folgende Vorgehensweise ist von Microsoft für die Installation des SDKs beschrieben:

- Installation WIN NT 4.0 Server oder Workstation englisch
  einfache Standardinstallation
- Installation SP3 englisch
- Installation IE4.01 englisch
  einfache Standardinstallation ohne Active Desktop
- Installation Optionpack für Workstation englisch mit IIS4.0 (Optional)
  einfache Standardinstallation des Optionpacks
  Die Installation des Optionpack mit IIS4.0 ist nur bei Benutzung des WBEM HTML Viewers nötig. Durch diese Komponente werden die Klassendefinitionen und Instanzen im CIM Repository als HTML Seiten dargestellt.
- Installation WBEM SDK
  Die Installation wird mit einem Doppelklick auf die Datei "WBEMSDK.EXE" gestartet. Folgendes Bild erscheint:

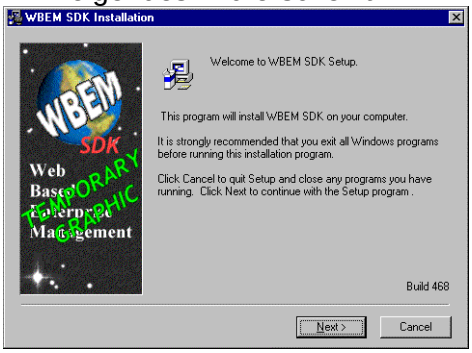

Abbildung A-1: MS WBEM SDK Installation Begrüßungsbildschirm

Durch einen Klick auf die Schaltfläche "NEXT >" wird die eigentliche Installation gestartet. Nach dem Akzeptieren des "License Agreement" erscheint folgendes Bild:

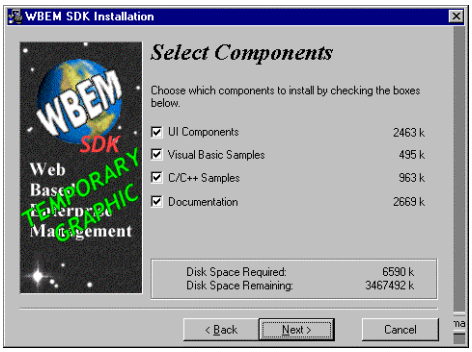

Abbildung A-2: MS WBEM SDK Installation Auswahl der Komponenten

Hier werden die einzelnen Komponenten des MS WBEM SDKs ausgewählt die Installiert werden sollen. Folgende Komponenten stehen zur Auswahl:

- UI Components
  Sämtliche Zusatzkomponenten wie CIM Studio und Event Viewer
- Visual Basic Samples
  alle in Visual Basic programmierten Beispiele
- C/C++ Samples
  alle in C/C++ programmierten Beispiele
- Dokumentation
  die Dokumentation des MS WBEM SDKs

Es wird empfohlen alle Komponenten auszuwählen
Durch Klicken auf die Schaltfläche "NEXT >" Startet das Kopieren und Registrieren der Komponenten. Das Installationsprogramm erkennt automatisch ob schon die sog. "Core Components" wie der CIMOM und das CIM Repository installiert sind. Falls nicht werden diese ebenfalls automatisch installiert.
Folgendes Bild erscheint:

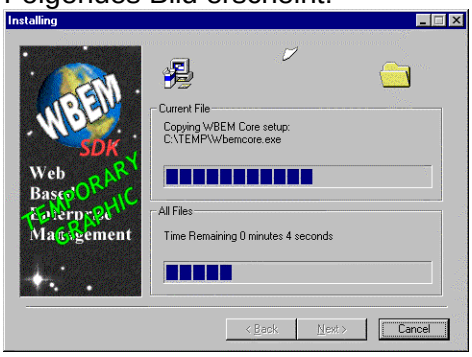

Abbildung A-3: MS WBEM SDK Installation Auswahl der Komponenten

Nach dem Kopieren der Dateien fragt das Installationsprogramm beim Benutzer mit der folgenden Dialogbox nach ob zur Installation des HTML Viewers der HTTP Server gestoppt werden kann. Diese Dialogbox erscheint nur wenn vorher das Optionpack mit dem HTTP Server installiert wurde.

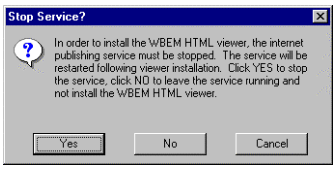

Abbildung A-4: MS WBEM SDK Installation Stop des HTTP Servers?

Nach der Installation des HTML Viewers ist die Installation des MS WBEM SDK abgeschlossen und folgendes Bild erscheint:

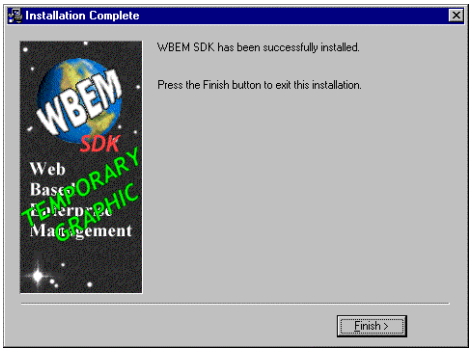

Abbildung A-5: MS WBEM SDK Installation Abschluß

Mit einem Klick auf die Schaltfläche "Finish >" ist die Installation abgeschlossen.

Bemerkungen:

Unsere Erfahrungen haben gezeigt, daß bei der Installation des Build 220 ein anderes als das vorgeschlagene Installationsverzeichnis anzugeben ist. Wird das vorgeschlagene Installationsverzeichnis benutzt, funktioniert zwar die Installation problemlos, aber jeglicher Zugriff auf das CIM Repository wird mit einer Zugriffsrechte-Verletzung abgewiesen, da das Build 220 anscheinend nur mit 8.3-Namen fehlerfrei umgehen kann.
Bei der Installation des Build 468 sind uns keine Fehler bekannt. Es fällt allerdings auf, daß ein Teil des SDKs immer im Verzeichnis WBEM unterhalb vom Verzeichnis „%Systemroot%\SYSTEM32“ installiert wird.

Abbildungsverzeichnis

Literaturverzeichnis

[SDK1] Dokumentation zum Microsoft WBEM SDK Beta2 Build 220

[SDK2] Dokumentation zum Microsoft WBEM SDK Beta2 Build 468

[CIM1] CIM V2.0 Shema Dokumentation http://www.dmtf.org/

[CIM2] CIM V2.0 Core Shema Dokumentation
ftp://ftp.dmtf.org/cim/CIM_CoreSchema20.PDF

[Todd] Greg Todd, What is WBEM? NT Magazine, Juli 1998, Seite 137

[FREE] http://wbem.freerange.com/

[MS1] Dokumente auf http://www.microsoft.com/management/wbem

[CeBit] Vortrag der Firma Compaq Computer auf der CeBit 1998

[PDC] Verschiedene Vorträge auf der Microsoft Profesional Developers Conference 1997, San Diego USA

[Fusion] Verschiedene Vorträge auf der Microsoft Fusion 1998, New Orleans USA

[TechEd] Verschiedene Vorträge auf der Microsoft TechEd 1998, New Orleans USA