FACULTEIT INGENIEURSWETENSCHAPPEN
EN ARCHITECTUUR

Besturingssystemen III

Bert De Saffel

Master in de Industriële Wetenschappen: Informatica Academiejaar 2018–2019

UNIVERSITEIT
GENT

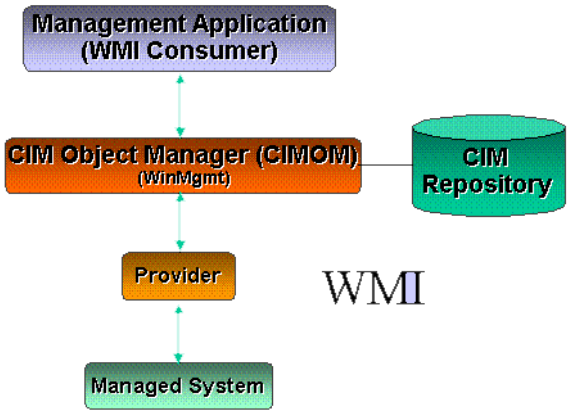
Inhoudsopgave

Hoofdstuk 1

WMI concepten

1.1 Herhaling SNMP en inleiding WMI

SNMP had de mogelijkheid om informatie bij te houden van devices die typisch op een netwerk aangesloten zijn zoals printers, routers, workstations en servers. Er was echter nood aan een systeem dat ook informatie van het hele operating systeem kon bevragen. Dit systeem noemt *Wbem*, een uniform systeem voor het bevragen van besturingssystemen zoals windows, HP en solaris. Een Windows specifieke versie hiervan is het *Windows Management Instrumentation* (WMI). Uit figuur 1.1 kunnen vijf componenten afgeleid worden:



Figuur 1.1: Basisarchitectuur WMI

- *WMI Consumer* : Dit zijn gebruikersapplicaties die gebruik maken van de CIMOM (= de WMI Service) om diverse informatie op te vragen. Deze kunnen in diverse programmeertalen geprogrammeerd worden (in dit vak wordt Perl gebruikt).
- *CIM Object Manager* : Wordt ook de WMI Service genoemd en heeft drie functionaliteiten:
  1. Interactie met de WMI Consumer op een **uniforme** manier.

2. De WMI Service is verantwoordelijk voor alle communicatie met de pr oviders. De WMI Consumer hoeft niets te weten over de verzameling Providers. De WMI Service gebruikt de CIM Repository om de juiste Provider te raadplegen.
3. De CIM Repository is enkel toegankelijk voor de WMI Service

- *CIM Repository*. Dit is een databank zowel waarbij software als hardware componenten als objecten voorgesteld worden. Deze object georiënteerde methode zorgt ervoor dat deze componenten via methoden kunnen aangesproken worden. Voor elke klasse wordt ook bijgehouden door welke provider deze klasse gerealiseerd wordt.
- *Provider* : Een provider is verantwoordelijk voor een aantal WMI-klassen
- *Managed System* : Dit zijn hardware componenten die aangesloten zijn aan een werkstation.

1.2 CIM Repository

De CIM Repository kan men inspecteren met een WMI-browser. Voorbeelden hiervan zijn *WMI CIM Studio* en *Powershell WMI Browser*. In deze samenvatting wordt gebruik gemaakt van WMI CIM Studio De inlogprocedure vraagt een namespace die standaard ingevuld staat op `root:CIMV2`. Deze namespace omvat ook de meeste WMI Objecten. Wat deze namespace niet heeft zijn klassen om office documenten zoals excel of word te manipuleren. Indien ingelogd, verschijnt er een scherm met twee panelen. Het linkerpaneel bevat alle klassen gesorteerd op hiërarchische volgorde op basis van overerving. De icoontjes naast elke klasse wordt gegenereerd op basis van de attributen van die klasse. Het rechterpaneel heeft standaard de *Properties* tab open. Dit tabblad bevat alle attributen van de geselecteerde klasse. Attributen voorafgegaan door 2 underscores zijn systeemattributen en bestaan voor elke klasse. Voorbeelden van zulke systeemattributen:

- __DERIVATION : Dit bevat een tabel met de hiërarchie van de overerving waarbij het eerste element de onmiddelijke superklasse is en het laatste element de root van de hiërarchie.
- __DYNASTY : Dit bevat de root van de hiërarchie van overerving. Deze informatie is dus dezelfde als het laatste element in het __DERIVATION attribuut.
- __PROPERTY_COUNT : Dit veld bevat een nummer met het aantal niet-systeemaanroepen
- __PATH : Dit veld bevat het absolute pad van de klasse.

Van elke klasse kan zijn instanties opgevraagd worden door op het icoontje rechts van het save-icoontje te klikken. Er wordt een twee dimensionale tabel weergegeven met per instantie een rij en per attribuut een kolom. Voor elke instantie moet het systeemattribuut __RELPATH aangevuld worden met de sleutel van het object. Dit is een string dat bestaat uit meerdere key-value paren dat toegevoegd wordt aan de default waarde van __RELPATH. Bij Singletons echter zijn unieke identificaties niet nodig en wordt __RELPATH aangevuld met een symbool. Een voorbeeld van een abstracte klasse en een singleton is respectievelijk `CIM_MonitorResolution` en `Win_LocalTime`.

1.3 Qualifiers

Een qualifier is een key-value koppel waarbij extra eigenschappen kunnen toegevoegd worden. Er bestaan vier soorten qualifiers:

1. klasse-qualifiers. Geeft extra informatie over de klasse.
2. attribuut-qualifiers. Geeft extra informatie over de attributen van een klasse
3. methode-qualifiers. Geeft extra informatie over de methoden van een klasse
4. methodeparameter-qualifiers. Geeft extra informatie over de parameters van de methoden van een klasse.

Klasse-qualifiers

Deze worden in WMI CIM Studio *Object qualifiers* genoemd:

- De **Description** qualifier geeft tekst terug met informatie over de klasse.
- De **Abstract**, **Dynamic** en **Singleton** qualifiers geven aan, indien hun waarde op *true* staat, dat een klasse abstract, dynamisch of een singleton is.
- De **Provider** qualifier helpt de WMI service te bepalen welke provider moet aangesproken worden om een bepaalde klasse aan te spreken

Attribuut-qualifiers

De attribuut-qualifiers worden in WMI CIM Studio *Property qualifiers* genoemd:

- De **Description** qualifier geef analoog zoals de klasse-qualifiers tekst terug met informatie over het attribuut.
- De **CIMType** qualifier geeft het type van het attribuut weer, zoals string, boolean, datetime, signed of unsigned int (8, 16, 32 of 64 bits) of referenties naar absolute objectpaden.
- De **Write** qualifier geeft aan, indien deze op *true* staat, dat de het attribuut rechtstreeks wijzigbaar is, zonder hiervoor methodes te moeten aanspreken.
- De **Keys** qualifier geeft aan dat het attribuut deel uitmaakt van de sleutel van het object en bijgevolg ook vermeld moet worden in het objectpad van een instantie.
- ValueMap geeft in een ééndimensionale tabel het domein weer van het attribuut: een expliciete opsomming van de toegelaten waarden. Values geeft een meer informatieve interpretatie van de toegelaten waarden. Indien ook de ValueMap qualifier aanwezig is, kunnen ValueMap en Values best beschouwd worden als respectievelijk de keys en de values van een Perl hash. Indien een ValueMap ontbreekt, dan impliceert Values een ValueMap met oplopende gehele getallen, startend vanaf 0. Values kan dan geïnterpreteerd worden als een Perl array.

Methode-qualifiers

- De **Description** qualifier geeft uitleg over de methode.
- De **CIMType** qualifier specifieert het return type.
- De **ValueMap** en **Values** worden vaak gebruikt bij een integer returnwaarde om deze waarde te vertalen naar iets zinvols.

- De **Privileges** qualifier specifieert een lijst van privileges die nodig zijn om de methode op te roepen. Indien deze qualifier niet aanwezig is kan eender wie de methode uit voeren.

De bijzonderste methode is de *Create* methode. Dit is een statische methode die onafhankelijk is van een specifieke instantie.

Een voorbeeld van een klasse met enkel statische methoden is **StdRegProv**. Deze klasse bevat methoden die het register kan aanpassen.

Methodeparameter-qualifiers

- De Description, CIMType, Values en ValueMap qualifiers hebben een analoge functie als de corresponderende attribuut- en methodequalifiers, nu toegepast op een individuele parameter van een methode.
- **In/Out** geeft aan dat deze parameter een invoer- (resp. uitvoer-) parameter is. Een parameter kan ook invoer/uitvoer parameter zijn.
- De **Optional** qualifier, indien ingevuld op true, geeft aan dat deze parameter optioneel is. Voor elke optionele qualifier is er wel een default waarde.
- **ID** geeft de volgorde aan (startend met de waarde 0) waarin de parameters als argumenten bij de methodeaanroep moeten opgegeven worden.

1.4 Associatorklassen

Stel dat je informatie wenst over de ethernet adapter. De klasse **Win32_NetworkAdapter** klasse kan allesinds de ethernet adapter filteren, maar er ontbreekt informatie zoals het IP adress. Die informatie bevindt zich in een andere klasse, **Win32_NetworkAdapterConfiguration**. In beide klasse zijn er echter geen verwijzingen naar de andere klasse. Voor zowel 1:1, 1:n en m:n relaties gebruikt WMI associatorklassen. Dit zijn klassen met meestal 2 attributen die een klasse met een andere klasse associeert. Eén van de instanties van de associatorklasse **Win32_NetworkAdapterSetting** zal in het eerste attribuut *element* een verwijzingen hebben naar de ethernet adapter in Win32_NetworkAdapter en zal in het tweede attribuut *setting* een verwijzing hebben naar de ethernet adapter in Win32_NetworkAdapterConfiguration zodat deze informatie gecombineerd kan worden.

1.5 WMI Query Language

WMI Consumers kunnen gebruik maken van **WQL**, een querytaal gebaseerd op SQL waarbij enkel de `SELECT`, `FROM` en `WHERE` clausule beschikbaar zijn. Clausules zoals `GROUP BY`, `HAVING` en `JOIN` zijn *niet* mogelijk in WQL. Het opvragen van alle ethernetadapters wordt bereikt met volgend stukje WQL code:

```
select *
from win32_networkadapter
where netconnectionid = 'Ethernet'
```

Attributen die gebruikt worden zijn niet hoofdlettergevoelig. Query's kunnen uitgetest worden in het programma **wbemtest.exe**. Bij het opstarten moet er eerst geconnecteerd worden naar een namespace dat standaard ingevuld is op *root\cimv2*. Indien op de knop *Query* geklikt wordt zal er

een venster verschijnen waarin WQL queries kunnen uitgevoerd worden. Het is ook mogelijk om de instanties op te vragen die gelinkt zijn aan het doelobject via associatorklassen.

```
associators of {win32_directory.name="C://..."}
where resultclass=win32_directory
resultrole=partcomponent
```

Verschillende predicaten in de *where* clausule worden niet verbonden met het AND keywoord zoals men verwacht, deze wordt namelijk impliciet ingevuld.

1.6 Notification queries

Een WMI Consumer kan zich aanmelden aan de WMI Service indien deze een interesse heeft voor een bepaalde gebeurtenis. De provider weet welke consumers geïnteresseerd en zal dan ook de consumers inlichten wanneer de voorwaarden voldaan zijn. Een consumer hoeft hier zelf geen processortijd aan te spenderen. Bij het uittesten van notification queries staat de optie *asynchronous* best in *Wbemtest* aan, anders is de uitvoeringstijd langer. Een voorbeeld van een notification query:

```
select *
from win32_processtrace
where processname = "calc.exe"
```

Deze query zal de consumer inlichten wanneer een windows rekenmachine opgestart of afgesloten wordt met een **Win32_ProcessTrace** object. In een notification query kan ook geaggregeerd worden. Deze komt zo goed als overeen als de SQL group by clausule, behalve dat er nog een extra sleutelwoord komt: `WITHIN`. Een `WITHIN` extensie specifieert het interval in seconden wanneer er geaggregeerd moet worden. Er is verder ook nog een `HAVING` clausule, die exact dezelfde functie heeft als bij SQL. Volgend voorbeeld zal om de 5 seconden enkel geaggregeerde informatie tonen indien meer dan 5 processen van het type *calc.exe* of *notepad.exe* aangemaakt of gesloten worden. Eerst wordt er geaggregeerd op de processnaam, daarna wordt er nog onderscheidt gemaakt tussen het type klasse. Dit is ofwel **Win32_ProcessStartTrace** of **Win32_ProcessStopTrace**.

```
select *
from win32_processtrace
where (processname = "calc.exe"
    or processname = "notepad.exe)
group within 5 by processname, __CLASS
having numberofevents >= 5
```

De geaggregeerde informatie komt in een object __AggregateEvent. Zo een object heeft slechts twee attributen:

1. *NumberOfEvents*: Dit geeft aan hoeveel events er zijn afgevuurd binnen het interval van de within extensie.

2. *Representative*: Dit attribuut verwijst naar één van deze gebeurtenissen. Informatie over de andere events gaan verloren.

Een spijtige zaak is dat notification queries maar voor een beperkt aantal providers mogelijk is. Indien eventobjecten voor een bepaald object niet bestaan moet men beroep doen op het **snapshotmechanisme**. In de query komt er een `WITHIN` extensie, die functioneel verschillend is van de `WITHIN` extensie van de group by. Deze nieuwe `WITHIN` extensie zal een interval in seconden aangeven

waarop een nieuwe snapshot moet gemaakt worden. Verder is er nog de `ISA` operator. Deze operator laat toe om een variabele met een klasse te vergelijken.

Volgend voorbeeld illustreert dezelfde query, zonder de group by clausule en met behulp van de **__InstanceOperationEvent** klasse, om events te genereren bij het aanmaken of vernietigen van processen met de naam calc.exe en notepad.exe.

```
select *
from __instanceoperationevent
within 2
where (__CLASS = "__instancecreationevent"
    or __CLASS = "__instancedeletionevent")
and targetinstance isa 'win_32_process'
and (targetinstance.name = "calc.exe"
  or targetinstance.name = "notepad.exe")
```

Een belangrijke klasse is de **__TimerEvent** klasse. Deze eventklasse wordt gegenereerd door de klasse **__TimerInstruction**, die twee implementaties kent: **__AbsoluteTimerInstruction** en **__IntervalTimerInstruction**. Dit zijn ook direct twee van de weinige klassen waarvan je zelf instanties mag van maken. De klasse **__AbsoluteTimerInstruction** is niet zo interessant aangezien die maar één enkele event zal afvuren op een specifiek tijdstip. De **__IntervalTimerInstruction** klasse daarentegen, zal een event afvuren telkens het interval bereikt wordt. De klasse bevat twee belangrijke attributen: *TimerID*, een unieke identificatie van de timer en *IntervalBetweenEvents*, het interval in milliseconden tussen opeenvolgende events. Vanaf dat een instantie van een **__IntervalTimerInstruction** gemaakt wordt, kan via volgende notification query dit event onderschept worden

```
select *
from __timerevent
where timerid=...
```

waarbij ... vervangen wordt door het gewenste *TimerID*. In het slechtste geval kan dit ook dienen als pollingsmechanisme, als snapshots ook niet werken.

1.7 Consumerprogrammas

Windows voorziet een aantal consumerprogrammas. Twee interessante zijn **SMTPEventConsumer** en **CommandLineEventConsumer**.

1.7.1 SMTPEventConsumer

Zoals de naam het doet vermoeden, zal deze consumer een mail sturen bij het afvuren van een specifiek event. Ten eerste moet er een instantie gemaakt worden van de klasse **__EVENTFILTER**. Belangrijke attributen zijn: *QueryLanguage*, dat best ingesteld wordt op WQL, *query*, een WQL query en *name*, een unieke identificatie. Nu kan een instantie gemaakt worden vand e klasse **SMTPEventConsumer**. Deze klasse heeft de volgende belangrijke attributen:

- *Name*: een unieke identificatie.
- *ToLine*: het e-mailadres van de ontvanger.
- *FromLine*: het e-mailadres van de vertegenwoordiger van de mail.

- *Subject*: het onderwerp van de mail.
- *Message*: het bericht van de mail.
- *SMTPServer*: een smtpserver, bv die van Google.

Deze twee klassen moeten nog aan elkaar gelinkt worden. Dit wordt gerealiseerd met behulp van de associatorklasse **__FilterToConsumerBinding**. Het attribuut *Filter* moet het __PATH attribuut bevatten van de gewenste filterinstantie. Het attribuut *Consumer* moet het __PATH attribuut bevatten van de gewenste consumerinstantie.

Stel dat toegepast moet worden met volgende query:

```
select *
from win32_processtrace
where (processname = "calc.exe"
or processname = "notepad.exe)
group within 5 by processname, __CLASS
having numberofevents >= 5
```

Het *query* attribuut van de **__EventFilter** klasse moet ingevuld worden met deze query. De attributen van **SMTPEventConsumer** kunnen als volgt ingevuld worden:

```
Name=mail1
ToLine=bert.desaffel@gmail.com
FromLine=bert.desaffel@gmail.com
Subject=%Representative.__CLASS% %NumberOfEvents%
Message=%Representative.Name%
SMTPServer=smtp.ugent.be
```

1.7.2 CommandLineEventConsumer

Deze consumer zal een command line commando uitvoeren na het afvuren van een specifiek event. De configuratie gebeurd op dezelfde manier als bij een **SMTPEventConsumer** instantie. Volgend voorbeeld gebruikt een **__EVENTFILTER** instantie waarbij de query als volgt ingesteld is:

```
select *
from win32_processstarttrace
where processname = 'calc.exe'
```

De **CommandLineEventConsumer** wordt als volg geïnstantieert. Dit zal het proces dat net opgestart wordt, terug killen.

```
Name=command1
CommandLineTemplate=taskkill /f /pid %ProcessID%
```