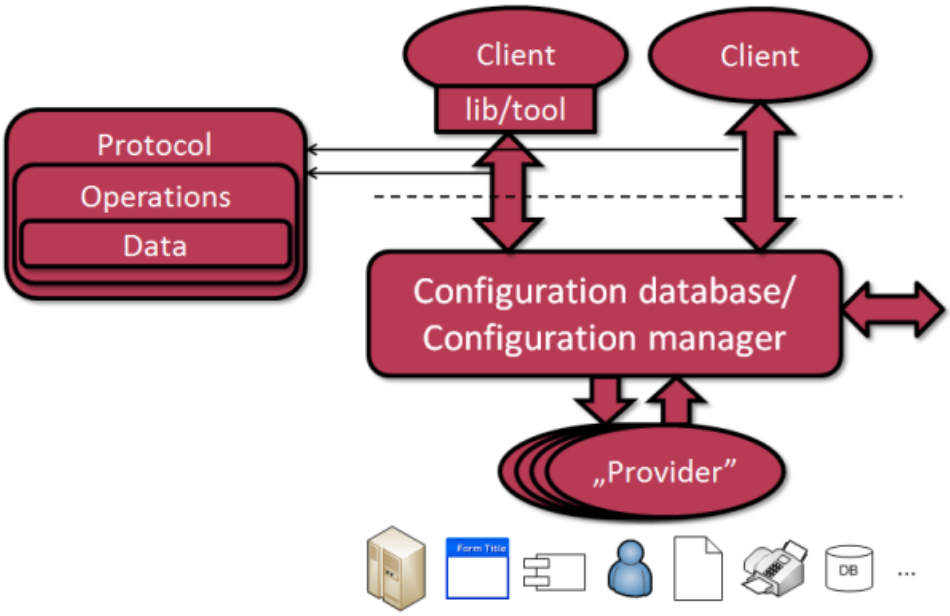
Konfigurációkezelési technológiák

Gyakorlati útmutató
Készítette: Micskei Zoltán
Utolsó módosítás: v1.1.3, 2012.04.09.

A segédlet célja, hogy bemutassa a konfigurációkezelési technológiákhoz tartozó alap eszközöket, és hogy megismerkedjünk a *Common Information Model*lel (CIM), a kapcsolódó szabványokkal (pl. WS-Management, CIM-XML) és ezek implementációival.

Az általános konfigurációkezelési ábránk a következő:



1. ábra: Konfigurációkezelés általános architektúrája

Figyelem:

- Az utasításokat ne másoljuk, hanem tényleg gépeljük is be. Különben nem sok mindent tanulunk belőle, nem rögzül a szintaktika.
- A gyakorlat elvégzése előtt nézzük át a kapcsolódó két előadást!

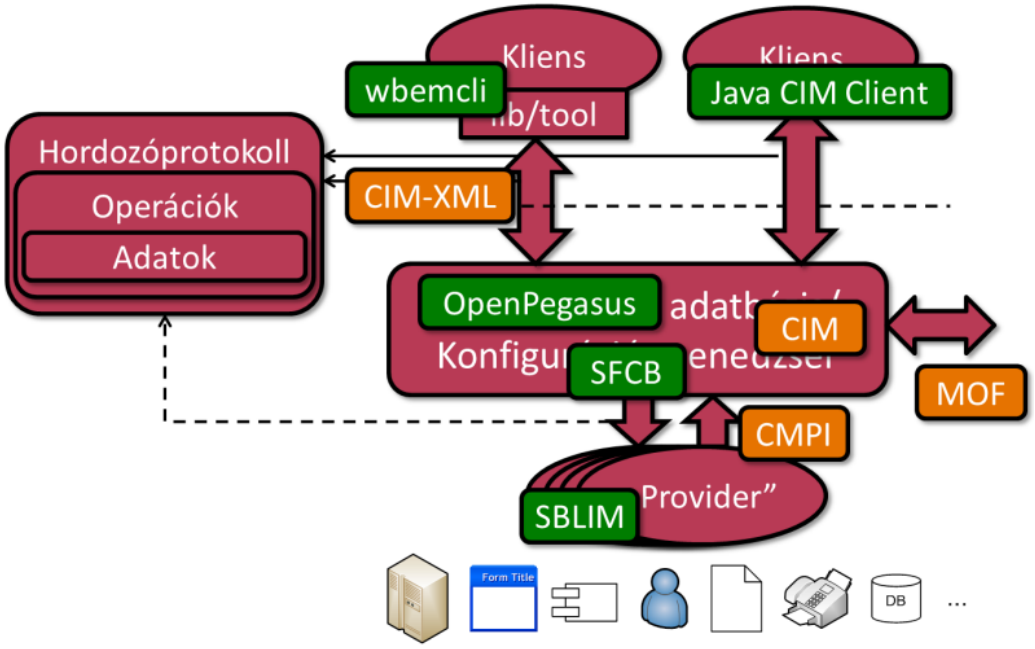
Tartalom

1 Linux: sblim-sfcb, wbemcli és openwsman

A feladatok megoldásához például a kiadott VMware virtuális gépbe telepített CentOS rendszert lehet használni. Ez a virtuális gép előre telepítve tartalmazza a következőket:

- CIM Object Manager: sblim-sfcb[1] 1.3.11-2
- CIM-XML kliens: wbemcli[2] 1.6.2
- Providerek: sblim-cmpi-*
- WS-Management szerver: openwsman[3] 2.3
- WS-Management kliens: wsmancli 2.2.7.1

Az általános konfigurációkezelési ábránkat tehát a következő módon fedik le az eszközök és az általuk felhasznált szabványok.



2. ábra: Konfigurációkezelő technológiák Linuxon

Megjegyzés: a leírásban szereplő elérési utak arra az esetre vonatkoznak, ha csomagkezelővel telepítjük fel a programokat. Ha forrásból fordítjuk, akkor alapesetben a `/usr/local` prefixet hozzárakja a legtöbb program a fájlok elérési útjához.

[1] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Sfcb
[2] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Wbemcli
[3] Web: http://openwsman.github.com/,
RPM csomagok: http://download.opensuse.org/repositories/systemsmanagement:/wbem/

1.1 sblim-sfcb

Az első feladatban áttekintjük az *Small Footprint CIM Broker (sfcb)*, a *CIM Object Manager* (CIMOM) főbb beállításait.

1. Indítsuk el a virtuális gépet!

   A gépre SSH segítségével lépjünk be (így pl. ki lehet másolni adatokat a kimenetről).

2. sfcb beállításai

   Az sfcb konfigurációs állományai a `/etc/sfcb` könyvtárban vannak. (A *pem* kiterjesztésű fájlok digitális tanúsítványok.)

   - Milyen állományokat találunk itt, ezek mit tárolnak?

   Keressük ki a beállításokat tároló fájlból, hogy a https engedélyezve van-e!

   ```
   nano /etc/sfcb/sfcb.cfg
   ```

   Az `sfcb.conf` fájl részletesen van kommentezve, de a beállítások leírását megtaláljuk az `sfcbd` manual oldalán is.

3. sfcb elindítása

   Kérdezzük le, hogy fut-e a CIM kiszolgáló:

   ```
   sudo /etc/init.d/ sblim-sfcb status
   ```

   Ha fut, akkor visszaadja az elindított folyamatok azonosítóját (PID). Ha nem fut, akkor indítsuk el:

   ```
   sudo /etc/init.d/sblim-sfcb start
   ```

   Ez így a legtöbb esetben megfelelő, azonban ha hibakeresésnél kíváncsiak vagyunk az sfcb üzeneteire is, akkor indítsuk azt is az előtérben. A `-t` megadásával még részletesebb trace üzeneteket kaphatunk az egyes komponensektől, a `-k` pedig komponensenként színezi az üzeneteket. A lehetséges komponensek listája:

   ```
   sfcbd -t ?
   ```

   (Elég sok üzenetet lehet így generálni, úgyhogy érdemes csak a szükségeseket megadni.)

4. A CIMOM által használt port kiderítése

   Nézzük meg, hogy milyen portokon figyel jelenleg a virtuális gép (a `-t` a TCP kapcsolatokat jeleníti meg, a `-l` a listening állapotban lévőket, a `-p` a hozzá tartozó folyamatot keresi ki):

   ```
   sudo netstat -t -l -p
   ```

- Keressük ki az `sfcbd` folyamathoz tartozó port számát (a --numeric hatására numerikus formában jeleníti meg az ismert portokat is, különben a `/etc/services` fájlban találjuk meg a megfeleltetést).

5. Az sfcb CIM tárhelye (repository)

   Az sfcb a `/var/lib/sfcb/registration` könyvtárban tárolja az általa ismert CIM osztályokat, az osztályokhoz tartozó névtereknek megfelelő alkönyvtárakban. Nézzük meg a tárhely tartalmát:

   ```
   cd /var/lib/sfcb/registration
   ls -R
   ```

   Alapesetben a `root/cimv2` és a `root/interop` névtereket szolgálja ki az sfcb, a CIM osztályok definícióit pedig `classSchemas` nevű bináris fájlokban tárolja.

   A tárhely tartalmát csak offline módon, az sfcb leállított állapotában lehet módosítani. Ilyenkor először egy ideiglenes, úgynevezett `stage` könyvtárba kell bemásolni a CIM osztályok definícióját tartalmazó MOF fájlokat, majd a stage tárhelybe kell importálni ezeket (a MOF állományok „lefordításával"). A `stage` könyvtárban a MOF osztálydefiníciókon kívül még szükség van úgynevezett provider regisztrációs fájlokra, ezek mondják meg, hogy melyik CIM osztályhoz milyen fájlban található az osztály példányait szolgáltató provider megvalósítása.

   Nézzük meg a `stage` könyvtár tartalmát:

   ```
   ls /var/lib/sfcb/stage
   ls /var/lib/sfcb/stage/mofs/root/cimv2
   ```

   Nézzük meg az egyik MOF fájl tartalmát:

   ```
   cat /var/lib/sfcb/stage/mofs/root/cimv2/Linux_Base.mof
   ```

   Nézzük meg az egyik regisztrációs fájl tartalmát is:

   ```
   cat /var/lib/sfcb/stage/regs/Linux_Base.reg
   ```

   Keressük ki az összes osztálynevet a MOF állományokból:

   ```
   cat /var/lib/sfcb/stage/mofs/root/cimv2/*.mof | grep ^class
   ```

   Az sfcb ezen kívül még felhasználja az alap CIM osztályok definícióját, ezt a következő könyvtárban találjuk:

   ```
   ls /usr/share/mof/cim-current/
   ```

   - A DMTF hivatalos CIM sémájának milyen verzióját tárolja jelenleg a gép?

   Ahhoz, hogy a providerek ténylegesen tudjanak is adatokat szolgáltatni, még egy dologra van szükség, ez pedig a providerek megvalósítása. Az sfcb ehhez a CMPI

specifikációnak megfelelő providereket használ, ezeket a következő könyvtárban tárolja:

```
ls /usr/lib/cmpi/
```

Ezzel nagyjából képet kaphatunk arról, hogy milyen formában kellene egy új provider és a hozzá tartozó CIM osztály definícióját megadni. Az sfcb adattárát az `sfcbrepos` parancs segítségével lehetne újraépíteni, de ezt, hacsak nincs valami nagyon nagy gond, ne adjuk ki.

A fejezetben leírtakról további információ az sfcb parancsaihoz tartozó manual oldalakon található, vagy pedig az sfcb projekt README állományában[4].

1.2 Helyi lekérdezések

A következő fejezetben a `wbemcli` parancsot fogjuk használni, hogy a CIM-XML protokoll segítségével pár egyszerűbb lekérdezést végrehajtsunk helyileg.

1. Alap osztály lekérdezése helyben

   Kérdezzünk le egy olyan osztályt, ami biztos létezik:

   ```
   wbemcli gc 'http://localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   A `gc` a `GetClass` művelet rövidítése.

   Az előbb a `netstat` kimenetéből kiderült, hogy például az 5989-es porton (ez a wbem-https port) és az 5988-as porton is figyelünk (ez a wbem-http port). Amíg tesztelünk és hibát keresünk, érdemes a http portot használni, hogy például Wiresharkban meg tudjuk nézni a forgalmat, azonban ha összeállt már a rendszer, érdemes kipróbálni https használatával is, éles rendszerben annak a használata a javasolt.

   Az objectPath-ban meg kell adni a CIMOM elérési információit (protokoll, DNS név vagy IP-cím, port), a névteret (`root/cimv2`), majd egy kettőspont után az osztály nevét (`CIM_OperatingSystem`).

   A válasz erre az, hogy hitelesíteni kell magunkat:

   ```
   *
   * wbemcli: Http Exception: Username/password required.
   *
   ```

   Adjunk meg felhasználónevet és jelszót is:

   ```
   wbemcli gc 'http://meres:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   Ezt se fogadja még el:

   ```
   *
   * wbemcli: Http Exception: Invalid username/password.
   *
   ```

[4] http://sblim.cvs.sourceforge.net/sblim/sfcb/README?view=markup

Az sfcb azoknak a felhasználóknak enged hozzáférést, akik benne vannak az `sfcb` nevű csoportban. Adjuk most hozzá a `meres` felhasználót ehhez a csoporthoz (Figyelem: ennek komoly biztonsági következményei vannak, ez a felhasználó innentől kezdve nagyon sok mindent le tud kérdezni és be tud állítani a rendszeren!):

```
sudo usermod -G -a sfcb meres
```

Ha most megismételjük, akkor most már sikeres a lekérdezés.

Hogy a kimenet olvashatóbb legyen, adjuk meg az `-nl` paramétert (new line, új sorokat szúr be az egyes tulajdonságok után):

```
wbemcli -nl gc 'http://meres:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
```

Még annyit nézzünk meg, hogy a háttérben milyen üzeneteket küld és kap a CIM-XML kliensünk. Erre a `-dx` kapcsoló való.

- Keressük ki a kapott üzenetekből, hogy a `TotalSwapSpaceSize` tulajdonság milyen típusú!

Kérdezzük le az osztály példányait is. Erre az `ei`, `EnumerateInstances` művelet használható.

```
wbemcli -nl ei 'http://meres:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
```

- A legelső sor a példány teljes nevét tartalmazza, ebben benne vannak a kulcs attribútumai és azok értékei. Keressük ki, hogy ennél az osztálynál mik a kulcs attribútumok!
- A kimenetben keressük meg, hogy mikor bootolt fel az operációs rendszer (LastBootUpTime tulajdonság)!

2. Egy adott példány lekérdezése

Ha csak egy adott példányra vagyunk kíváncsiak, akkor azt a `gi`, `GetInstance` művelet használható. Ebben az esetben az objektum elérési útjában meg kell adni a kulcs attribútumainak az értékét is:

```
wbemcli -nl gi
'http://meres:LaborImage@localhost:5988/root/cimv2:Linux_OperatingSystem.Name="ir
fserver.irf.local",CreationClassName="Linux_OperatingSystem",CSName="irfserver.ir
f.local",CSCreationClassName="Linux_ComputerSystem"'
```

Ha nem adnánk meg kulcsokat, akkor a következő hibaüzenetet kapjuk:

```
*
* wbemcli: Cmd Exception: Keys not specified
*
```

Ha nem az összes kulcsot adjuk meg, akkor az osztálytól függően változatos hibaüzeneteket kapunk eredményül. Ilyenkor érdemes pl. a következő fejezetben ismertetett ECUTE segítségével ellenőrizni, hogy mik az adott osztály kulcsai.

3. Lekérdezhető osztályok listája

   Nézzük meg, hogy a rendszer milyen CIM osztályok definícióját tárolja (figyelem, ez nem feltétlenül jelenti azt, hogy van is példány mindegyikhez):

   ```
   wbemcli -nl ecn 'http://meres:LaborImage@localhost:5988/root/cimv2'
   ```

4. Lekérdezés https felületen

   Próbáljuk meg az előbbi kérést végrehajtani úgy, hogy egy SSL csatornát építünk ki a kapcsolathoz. Ehhez a protokollt és a portot is meg kell változtatni:

   ```
   wbemcli -nl ecn 'https://meres:LaborImage@localhost:5989/root/cimv2'
   ```

   A válasz erre a következő hibaüzenet:

   ```
   *
   * wbemcli: Http Exception: Peer certificate cannot be authenticated with known CA
   certificates
   *
   ```

   Tehát az sfcb kiszolgáló által felkínált tanúsítványt a helyi gépünk nem fogadta el hitelesnek, mert nem egy megbízható hitelesítésszolgáltató (certificate authority – CA) állította ki. Ilyenkor a következőket tehetjük:

   - Megmondjuk a `-noverify` kapcsolóval a wbemcli eszköznek, hogy ne is akarja ellenőrizni a tanúsítványt.
   - Egy hiteles tanúsítványt állítunk be az sfcb kiszolgálónak (ez egy tesztrendszer esetén nem túl reális opció).
   - A `-cacert` kapcsoló segítségével megadhatjuk annak a CA-nak a tanúsítványát, aki kiállította az sfcb kiszolgáló tanúsítványát vagy a kiszolgáló self-signed tanúsítványát.

   Vigyázat: a kiadott virtuális gépen használt verzióknál ha root felhasználóként futtatjuk a wbemcli programot és https protokollt használunk, akkor segmentation fault lesz az eredmény.

1.3 ECUTE

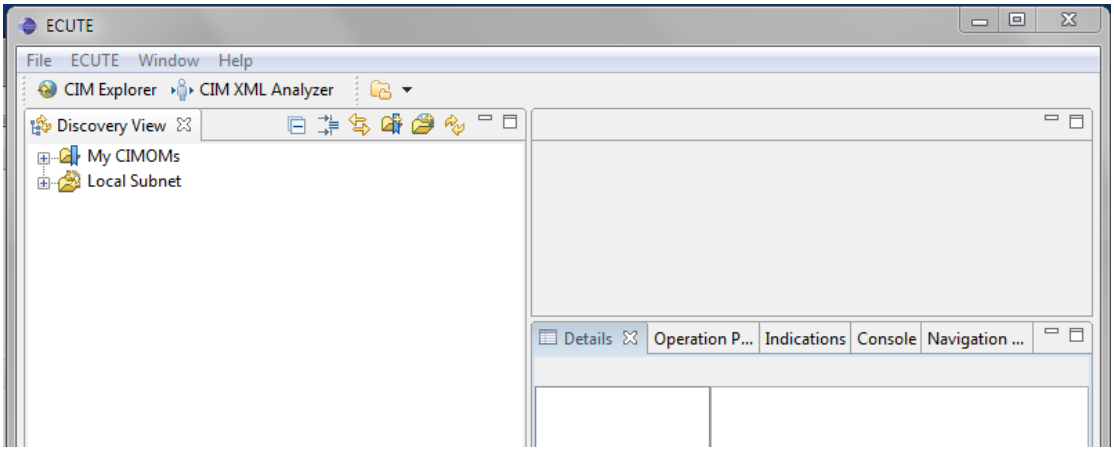
Az ECUTE (Extensible CIM UML Tooling Environment)[5] az SBLIM projekt része. Egy Eclipse alapú GUI-t biztosít CIMOM-ok megnézésére. Az adatokat CIM-XML segítségével kérdezi le. SLP (Service Location Protocol) segítségével fel is tudja deríteni, hogy milyen CIMOM-ok vannak a helyi hálózaton. A következő részekből áll:

[5] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Ecute

- *Explorer*: CIM osztályok és példányok adatainak lekérdezése.
- *Analyzer*: kommunikáció megfigyelésére és az üzenetek tartalmának megjelenítésére szolgáló komponens.
- *(Modeler)*: UML modellekből képes CIM leírásokat készíteni, de ehhez kell az IBM Rational Software Architect program is.
- *ECUTE Rich Client Platform*: ha a fenti komponenseket nem egy meglévő Eclipse példányban, hanem önálló alkalmazásként akarjuk futtatni, akkor ehhez kell az ECUTE RCP.

A telepítéshez töltsük le az ECUTE RCP-t, majd az Analyzer és Explorer tartalmát másoljuk be az RCP könyvtárába[6].

Elindítás után a következő kép fogad.



3. ábra: ECUTE képernyő

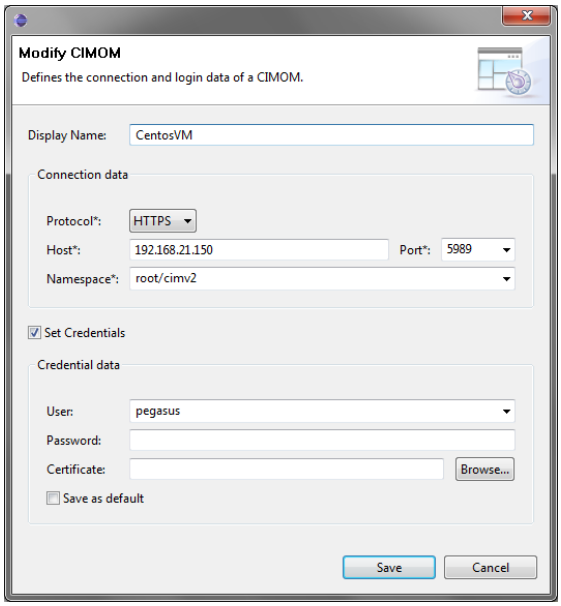
Az Explorer használatához először hozzá kell adni a *My CIMOMs* részhez egy új elemet (4. ábra).

Ezután már tudunk osztályneveket lekérdezni, az osztályok tulajdonságait megjeleníteni, valamint példányokat felsorolni. Az ECUTE súgója elég használható, ha elakadunk, azt érdemes megnézni.

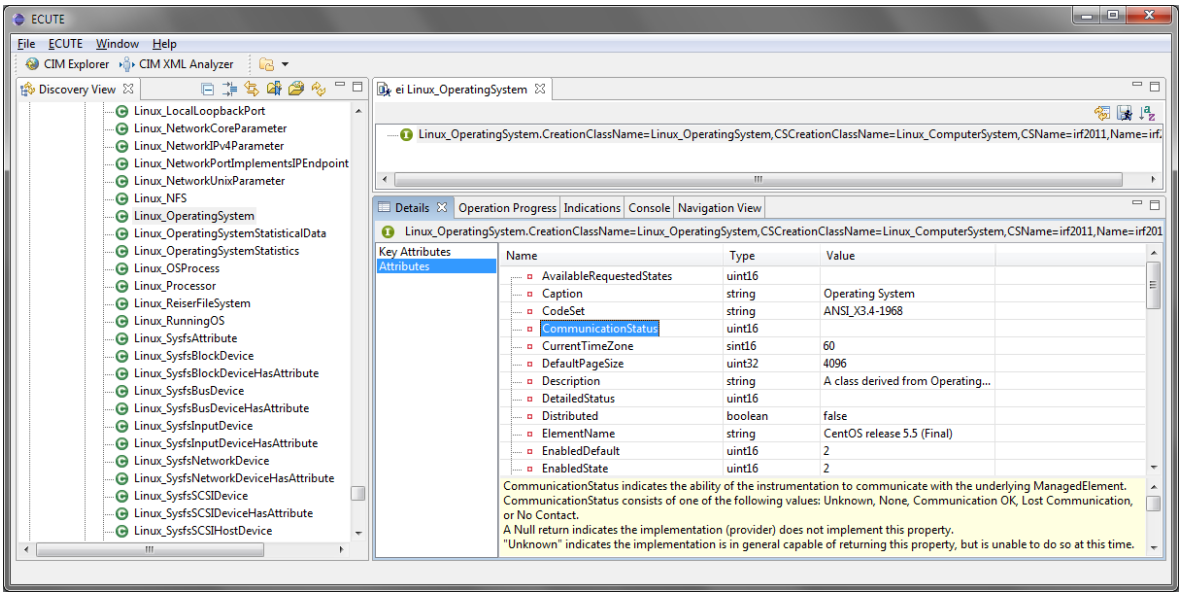
A következő képen a `Linux_OperatingSystem` példányait kérdeztük le, majd annak egy tulajdonságát nézzük meg (5. ábra).

Próbáljuk ki most az *Analyzer* komponenst is. Ehhez a CIMOM tulajdonságainál a kapcsolat típusát állítsuk át *Analyzer Connection* értékre (6. ábra).

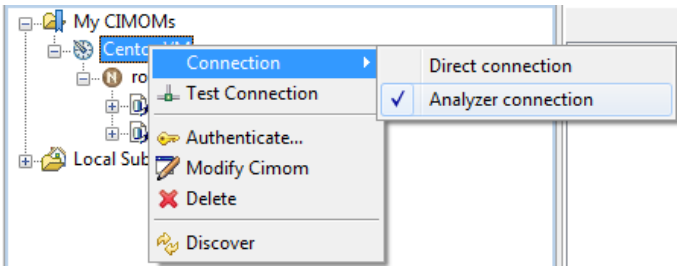
[6] 1.7-es Java használata esetén már nem biztos, hogy elindul az ECUTE RCP, mert annyira régi Eclipse verziót használ. Ilyen esetben egy újabb Eclipse-be kell telepíteni a letöltött Analyzer és Explorer feature-t (*Install New Software…* menüpont, majd *Local* kiválasztása az Update Site résznél), majd az *Open Perspective* menüben megtalálhatóak az ECUTE Explorer és Analyzer perspektívák.



4. ábra: CIMOM hozzáadása

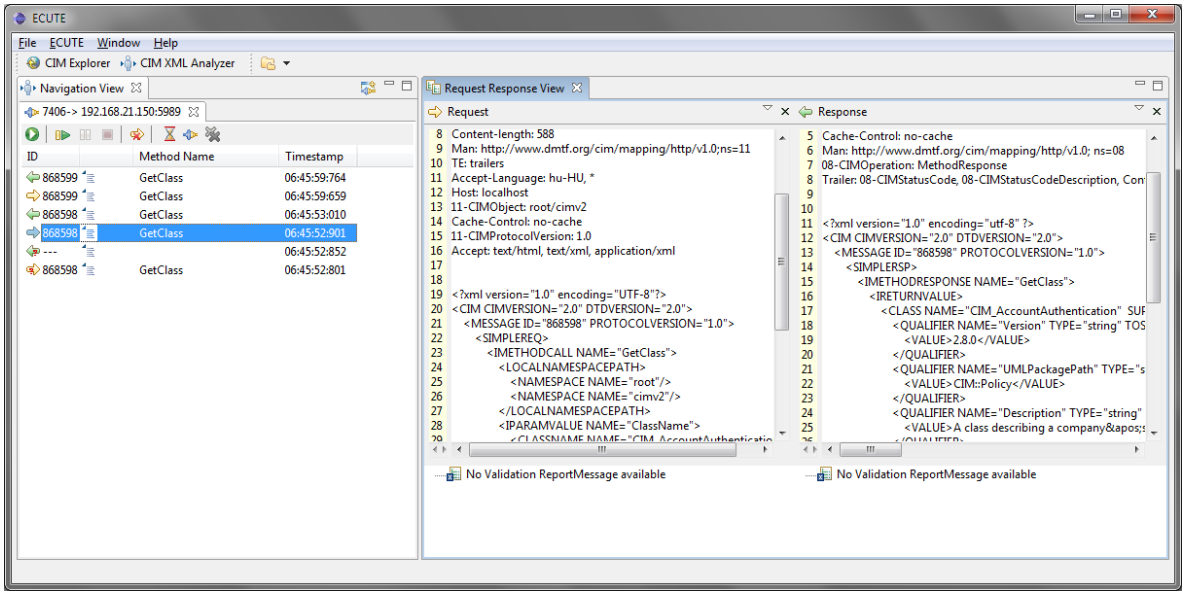


5. ábra: Lekérdezés az ECUTE-ban



6. ábra: Kapcsolat átállítása Analyzer módba

Innentől kezdve, ha végrehajtunk egy lekérdezést, akkor az az *CIM XML Analyzer* nézetben megjelenik, és a *Request Response View* nézetben meg is tudjuk nézni a tartalmát.



7. ábra: Analyzer nézet használata

Próbáljuk ki az ECUTE-ot:

- Nézzük meg a definiált CIM osztályokat, nézzük meg néhánynak a tulajdonságait.
- Kérdezzük le néhány osztály példányait. Figyelem, sok osztályhoz nincs megfelelő provider, ilyenkor „Not supported" választ fogunk visszakapni. (Tipikusan a Linux kezdetű osztályokhoz van provider).
- Kapcsoljuk be az Analyzert, és nézzünk meg egy-két konkrét CIM XML üzenetet, próbáljuk megtalálni benne, hogy milyen osztályt kérdezünk le.

1.4 Távoli lekérdezés wbemcli segítségével

Ahhoz, hogy távoli lekérdezéseket tudjunk végrehajtani, nyilván kell egy távoli gép is.

1. Távoli gép létrehozása

   Másoljuk le a CentOS virtuális gépet (kikapcsolt állapotban természetesen).

   A másolat indításakor válasszuk az *'I copied'* opciót, hogy biztos új MAC címet kapjon.

   Érdemes megváltoztatni a gép nevét, hogy az SSH ablakban és a konzolon ne keverjük össze a gépeket. Ehhez a következő fájlt kell megváltoztatni:

   ```
   /etc/sysconfig/network
   ```

   Ebben a `hostname` részt kell átírni. Ezen kívül érdemes még a `/etc/hosts` fájlban is megváltoztatni, hogy az új név is a `127.0.0.1` IP-címre feloldódjon, így elkerülhetőek a hosszú időtúllépések egyes programokban.

A változás után újra kell indítani a gépet (a `reboot` paranccsal).

2. Távoli lekérdezés futtatása

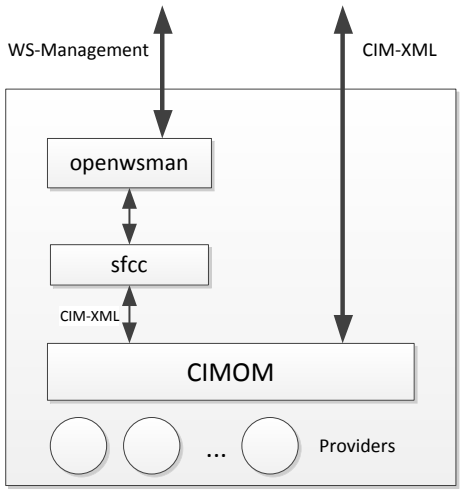
   Mivel már a helyi lekérdezések futtatásánál is megadtunk minden információt (protokoll, jelszó…), ezért most csak annyit kell tennünk a távoli lekérdezések futtatásához, hogy a `localhost` helyett a távoli gép IP-címét írjuk be.

   ```
   wbemcli gc 'http://meres:LaborImage@<tavoli_gep_ip_cim>:5988/root/cimv2:CIM_OperatingSystem'
   ```

   Nyilván ehhez az is kell, hogy a megfelelő port ki legyen engedve a tűzfalon. A kiadott virtuális gépen ki van kapcsolva a tűzfal, ez éles rendszerben nem engedhető meg.

1.5 openwsman

Az *openwsman* egy WS-Management protokollt megvalósító kiszolgáló. Önmaga konfigurációs adatokat nem kezel, hanem csak egy WS-Management interfészt biztosít egy meglévő CIMOM kiszolgálóhoz. (Tehát a CIMOM kiszolgálót az openwsmantól teljesen függetlenül kell beállítani, az openwsman egyszerűen csak CIM-XML kliensként csatlakozik hozzá.)



8. ábra: openwsman architektúrája

A fenti ábra szemlélteti az openwsman és a szükséges komponensek viszonyát (8. ábra). Az openwsman az SFCC[7] (Small Footprint CIM Client) könyvtárat használja, ami egy C nyelvű API-t ad CIMOM-ok elérésére CIM-XML-en keresztül. (Megjegyzés: az SFCC támogatja egy SfcbLocal nevű speciális kapcsolatot is, amivel az sfcb CIMOM-hoz tud gyorsabb módon csatlakozni).

1. openwsman kiszolgáló beállítása

   Az openwsman a beállításait a következő konfigurációs fájlban tárolja:

   ```
   /etc/openwsman/openwsman.conf
   ```

[7] http://sourceforge.net/apps/mediawiki/sblim/index.php?title=Sfcc

(Ha nem csomagkezelővel raktuk fel az openwsmant, hanem forrásból fordítottuk, akkor a fájl a `/usr/local/etc/openwsman/` könyvtár alá kerülhet.)

Nézzük meg a konfigurációs fájl tartalmát!

- Ha engedélyezve van, kapcsoljuk ki az IPv6 használatát!
- Milyen porton figyel a szerver?
- Milyen porton próbál meg csatlakozni a CIMOM-hoz?

A 2.3-as verziótól kezdve az openwsman Wiki oldalán[8] találhatóak információk a konfigurációs beállításokról. Régebbi verziókban az openwsman README fájlja elérhető volt tipikusan a `/usr/share/doc` alatt, a pontos helyét a `locate` segít megkeresni. Ez kiírja az összes olyan fájl teljes elérési útját, ami tartalmazza a megadott szöveget (a locate adatbázisát az `updatedb` paranccsal lehet frissíteni):

```
locate openwsman
```

Többféle hitelesítési módszert lehet alkalmazni, ez lehet GSS, HTTP Digest vagy Basic (a Digest használata már nem javasolt, Basic módszert pedig csak SSL-lel együtt érdemes éles környezetben használni). A felhasználói információkat tárolhatjuk sima password fájlokban vagy PAM (Pluggable authentication modules) segítségével az operációs rendszer felhasználói adatbázisát is használhatjuk (ez utóbbi a javasolt).

2. openwsman kiszolgáló futtatása

Nézzük meg, hogy milyen kapcsolói vannak az openwsman kiszolgálónak:

```
openwsmand --help
```

Indítsuk el debug módban, ilyenkor a konzol kimenetre írja ki a hibákat és információs üzeneteket.

```
sudo openwsmand –d
```

- Az openwsmand log üzenetei hosszúak, érdemes az SSH ablakot átméretezni, hogy a nagy részük egy sorba kiférjen.
- Hányféle művelet kezelőjét regisztrálta be az openwsman a kimenet alapján?

A következő paranccsal lehet háttérszolgáltatásként futtatni:

```
sudo /etc/init.d/openwsmand start
```

(Ehhez előbb be kéne konfigurálni az SSL-hez szükséges tanúsítványokat, ami a kiadott virtuális gépen nem történt meg.)

[8] https://github.com/Openwsman/openwsman/wiki

1.6 openwsman lekérdezése a wsmancli paranccsal

1. Ismerkedés a *wsmancli* paranccsal

   A WS-Management protokollhoz használhatjuk a *wsmancli* csomagban lévő `wsman` parancssori eszközt. Nézzük meg először a paraméterezését:

   ```
   wsman --help
   ```

   Így az alapvető kapcsolódáshoz szükséges paramétereket írja ki. A wsman ennél több paraméter kezelésére is képes (pl. event subscription), ezekre most a kezdeteknél nem lesz szükség. Ezeket egyébként a következő paranccsal tudjuk megnézni:

   ```
   wsman --help-all
   ```

   A wsman parancshoz nincs manual oldal, így néha kicsit nehézkes a paraméter pontos jelentését kitalálni. Ha a help üzenet nem segít, akkor pedig a wsman forrásában meg lehet nézni, hogy mit csinál az adott paraméterrel, a fő része egy darab C fájlból áll.

   Az általános formátum tehát a következő:

   ```
   wsman [Option...] <action> <Resource Uri>
   ```

   A kapcsolódási opciók után meg kell adni az akciót, amit végre akarunk hajtani, és esetlegesen az erőforrás URI-ját, amin az akciót el akarjuk végezni. A lehetséges akciók:

   ```
   get, put, create, delete, enumerate, pull, release, invoke, identify, anonid,
   subscribe, unsubscribe, renew, associators, references, test
   ```

2. Távoli fél azonosítása (identify)

   A legegyszerűbb művelet az IDENTIFY, ez csak lekérdezi a WS-Management kiszolgálót.

   A teszteléshez érdemes megnyitni két külön SSH munkamenetet, az egyikben futtathatjuk az openwsmand programot debug módban, a másikban pedig a wsman parancsokat. Paraméterként adjuk meg, hogy melyik géphez akarunk csatlakozni:

   ```
   wsman --hostname localhost identify
   ```

   (A paramétereknek van itt is rövid neve, a későbbiekben ezt fogjuk használni).

   Erre válaszként hitelesítést kér. Itt még bármelyik, a rendszerben létező felhasználót megadhatjuk, hisz még csak az openwsman kiszolgálóhoz csatlakozunk, ahol PAM segítségével a helyi felhasználói adatbázist használjuk, és nem adtunk meg semmi jogosultsági korlátot. Később, amikor már az openwsman segítségével a CIMOM kiszolgálót akarjuk lekérdezni, akkor olyan felhasználót kell majd megadni, akinek van joga a CIMOM-hoz is hozzáférni (a kiadott virtuális gépen ez alapesetben a `root` felhasználó, vagy a `meres`, ha az sfcb feladatoknál beállítottuk).

   Megadhatjuk a hitelesítési információkat paraméterként is:

   ```
   wsman --hostname localhost -u meres -p LaborImage identify
   ```

Így most már végre tudunk hajtani egy egyszerű kérést. Ha a `hostname` paramétert lecseréljük, akkor akár távolról is meg tudjuk szólítani az openwsman kiszolgálónkat.

A `wsmancli` 2.2.7-es verziója előtt az alapértelmezett portokat nem kezelte helyesen, ilyenkor a `-P 5985` kapcsolóval a portot kézzel kellett megadni.

Ha valami nem működne (ez sajnos az openwsman esetén nem ritka), akkor következik a hibakeresés. Fut-e az opewsmand? Írt-e ki hibát az elindulásakor? Hova is akarunk pontosan csatlakozni? Szerencsére a wsman is tud elég részletes debug üzeneteket is kiírni (`-d 6` kapcsoló), az sokszor segít.

3. CIM objektumok lekérdezése WS-Management segítségével

   Egy egyszerű lekérdezéshez meg kell adni egy műveletet és egy erőforrás URI-t (lásd a kapcsolódó előadásban). A művelet lekérdezéseknél GET vagy ENUMERATE, az erőforrás URI pedig egy névtér prefixből, osztálynévből és esetlegesen példány kiválasztókból (selector) áll.

   Nézzünk egy ENUMERATE műveletet valamelyik alap CIM osztályhoz. URI prefixként a DMTF által szabványosított URI-t lehet használni:

   ```
   http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2
   ```

   Az előző részben összerakott paraméterek alapján a lekérdezés így néz ki:

   ```
   wsman -h localhost -u meres -p LaborImage enumerate
   'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
   ```

   (Megjegyzés: az erőforrás URI köré rakható aposztróf vagy idézőjel, de ha nincs szóköz, egyéb speciális karakter vagy változó benne, akkor akár el is hagyhatóak ezek.)

   Az eredmény két XML dokumentum lesz. Az első egy *EnumerationContext* azonosítót ad vissza, amit utána `Pull` kérésekkel végiglépkedve megkapjuk a következő XML dokumentumokban a lekérdezés eredményét.

   ```
   ...
       <wsen:EnumerateResponse>
         <wsen:EnumerationContext>57f05744-a180-8005-17ee</wsen:EnumerationContext>
       </wsen:EnumerateResponse>
   ...
     <s:Body>
       <wsen:PullResponse>
         <wsen:Items>
           <n1:Linux_Processor>
             <n1:AddressWidth xsi:nil="true"/>
             <n1:Availability xsi:nil="true"/>
             <n1:CPUStatus>1</n1:CPUStatus>
   ...
   ```

   A `-O` paraméter megadásával kérhetjük azt, hogy a minden egyes XML dokumentumot külön fájlba mentsen el.

4. Egy konkrét objektum lekérdezése (GET művelet)

   Az előbbiekben egy adott osztály összes példányát kérdeztük le. Most nézzük meg, hogy hogyan lehet egy konkrét osztályt lekérdezni. Ehhez írjuk át az enumerate paramétert get-re:

   ```
   wsman -h localhost -u meres -p LaborImage get
   'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
   ```

   Válaszként egy *Fault* üzenetet kapunk, aminek a *Text* mezője tartalmazza a hiba okát:

   ```
   <s:Text>The Selectors for the resource are not valid.</Text>
   ...
   <wsman:FaultDetail>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/Insuffi
   cientSelectors</wsman:FaultDetail>
   ```

   A hiba teljesen jogos, hisz nem adtuk meg, hogy melyik konkrét példányt akarjuk lekérdezni. Ehhez kéne az úgynevezett kiválasztókat (selector) még hozzáfűzni az URI-hoz. Az osztály összes kulcsattribútumát meg kéne itt adni. Az, hogy melyek ezek, a legkönnyebben egy wbemcli-s lekérdezéssel deríthetjük ki:

   ```
   wbemcli -nl ei 'http://meres:LaborImage@localhost:5988/root/cimv2:CIM_Processor'
   ```

   Az elején kiírja a példány teljes nevét:

   ```
   localhost:5988/root/cimv2:Linux_Processor.CreationClassName="Linux_Processor",Dev
   iceID="0",SystemCreationClassName="Linux_ComputerSystem",SystemName="irfserver.ir
   f.localdomain"
   ```

   Ennek tehát négy darab kulcsa van és mindet meg kell majd adnunk a példány lekérdezéséhez:

   ```
   wsman -h localhost -u meres -p LaborImage get →
    'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor?→
   CreationClassName=Linux_Processor,DeviceID=0,SystemCreationClassName=Linux_Comput
   erSystem,SystemName=irfserver.irf.local'
   ```

   Az eredményként vissza is kapjuk a keresett példányt.

   Ha elírjuk az objektum attribútumainak az értékét, akkor a következő hibát kaphatjuk:

   ```
   No route can be determined to reach the destination role defined by the WS-
   Addressing To.
   ```

5. Nem DMTF szabványos osztály lekérdezése

   Az előző feladatban a wbemcli kimenetében láthattuk, hogy a lekérdezett `CIM_Processor` tulajdonképpen a `Linux_Processor` példánya. Próbáljuk meg ezt lekérdezni, ehhez a fenti parancsban csak az osztály nevét kell átírni. Az eredmény a következő hiba:

```
No route can be determined to reach the destination role defined by the WS-
Addressing To.
```

A probléma az, hogy nem jó névtér prefixet használtunk, hisz ezt az osztály már nem a szabványos CIM séma tartalmazza. Az openwsman konfigurációs beállításai között megtalálható, hogy milyen további plusz sémákat képes kiszolgálni, ezek között szerepel a Linux is:

```
Linux=http://sblim.sf.net/wbem/wscim/1/cim-schema/2
```

Használjuk tehát ezt a névteret:

```
wsman -h localhost -u meres -p LaborImage get
'http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_Processor?→
CreationClassName=Linux_Processor,DeviceID=0,SystemCreationClassName=Linux_Comput
erSystem,SystemName=irfserver.irf.local'
```

Legnagyobb megelégedésünkre így már működik ez a lekérdezés is.

Ezek alapján egy Linuxon futó CIMOM-ból tudunk helyben és távolról adatokat lekérdezni, mégpedig CIM-XML és WS-Management protokollon keresztül is.

6. Szűrés a lekérdezésben

Egy adott osztálynak lehet nagyon sok példánya is. Ha nem vagyunk az összesre kíváncsiak, akkor érdemes már a lekérdezésben szűrni. Nézzük meg például a következő lekérdezést:

```
wsman -h localhost -u meres -p LaborImage enumerate
http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_KernelParameter
```

Ennek az osztálynak a kiadott virtuális gépen 99 példánya volt, aminek a lekérése és a kiírása már észrevehető ideig tart. Ha például csak azokra vagyunk kíváncsiak, amiknek az `Edittable` tulajdonságának az értéke `false`, akkor megpróbálkozhatunk a következővel:

```
# the following is wrong, Edittable is not a key attribute
wsman -h localhost -u meres -p LaborImage enumerate →
 'http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_KernelParameter?→
Edittable="false"'
```

Azonban ilyenkor visszakapjuk az összes példányt, mert ez a szintaxis egy konkrét példány kiválasztására jó, és nem szűrésre. (A háttérben csak a SOAP üzenet `Header` részében a `wsman:SelectorSet` részt állítja be.)

Megjegyzés: hasonló esetben például a WinRM a következő hibát adná:

```
<f:Message>The following selector is not a key property of the resource accessed
: State.  Use selectors that are key properties for the resource that you want to
access. </f:Message>
```

Szűrésre a `--filter` tulajdonság való, ehhez be kell állítani a `--dialect` paraméterrel a szűrés dialektusát is. A dialektust a háttérben lévő CIMOM-nak kell támogatnia, az openwsman csak továbbítja a kérést. A dialektusokat egy URI azonosítja, a tipikusak:

```
Selector      http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter
Association   http://schemas.dmtf.org/wbem/wsman/1/cimbinding/associationFilter
CQL           http://schemas.dmtf.org/wbem/cql/1/dsp0202.pdf
XPath         http://www.w3.org/TR/1999/REC-xpath-19991116
WQL           http://schemas.microsoft.com/wbem/wsman/1/WQL
```

A Selector dialektus esetén egyszerűen az adott osztály attribútumainak értékeire lehet feltételeket megfogalmazni, konkrét értékek ÉS kapcsolata szerepel a szűrőben. Lássunk egy példát rá:

```
wsman -h localhost -u meres -p LaborImage enumerate →
 http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_KernelParameter →
 --dialect="http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter" →
 --filter="Edittable=false"
```

Ez visszaadja (visszaadná) azokat a példányokat, ahol az Edittable tulajdonság értéke false[9]. (A példában látszik, hogy a wsman parancs nem válogatós, és az opciókat elfogadja a ResourceURI után is.)

Lehet több attribútum értékére vonatkozó feltételt is megfogalmazni:

```
wsman -h localhost -u root -p LaborImage enumerate →
 http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_KernelParameter →
 --dialect=http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter →
 --filter='Edittable=true,Value=2'
```

Ilyenkor a következő WS-Management kérés áll elő (ezt a wsman eszköz `-R` paraméterével tudjuk mi is megnézni):

```
<wsen:Enumerate>
  <wsman:Filter
      Dialect="http://schemas.dmtf.org/wbem/wsman/1/wsman/SelectorFilter">
    <wsman:SelectorSet>
      <wsman:Selector Name="Value">2</wsman:Selector>
      <wsman:Selector Name="Edittable">true</wsman:Selector>
    </wsman:SelectorSet>
  </wsman:Filter>
</wsen:Enumerate>
```

Arra figyeljünk, hogy a wsman eszköz jelenlegi verziójában mindig a vessző mentén darabolja a kifejezést, hiába van az esetleg idézőjelek között, így az érték nem tartalmazhat egyelőre vesszőt.

A Selector dialektus pontos definícióját a WS-Management szabvány [1] E függelékében találjunk.

[9] A 2.3-as openwsmanban még van egy bug, ami miatt csak egy példányt ad vissza, ha Selector dialektusú szűrést alkalmazunk.

Ha bonyolultabb lekérdezéseket akarunk végrehajtani, és a CIMOM a háttérben sfcb, akkor használhatjuk a CIM Query Language (CQL) dialektust. Ez egy SQL-szerű nyelv, amiben már kicsit bonyolultabb lekérdezéseket is meg lehet fogalmazni, például:

```
wsman -h localhost -u root -p LaborImage enumerate →
 http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_KernelParameter →
 --dialect="http://schemas.dmtf.org/wbem/cql/1/dsp0202.pdf" →
 --filter="SELECT SettingID, Value, Edittable FROM Linux_KernelParameter →
 WHERE Value='1' AND Edittable = true"
```

Ilyen esetben a WS-Management protokollban csak átadjuk a szűrőt, az openwsman továbbítja a CIMOM-nak, és az értelmezni. A fenti példában az Edittable attribútum típusa Boolean, míg a Value típusa String, ezért kell az egyik köré aposztróf és a másik köré nem. A CQL nyelv teljes definícióját további példákkal megtaláljuk a DMTF specifikációjában [3].

Ha CQL szűrőt használunk, és a szűrőben csak egy osztály szerepel a FROM után, akkor a ResourceURI lehet az adott osztály URI-ka. Ha több osztály szerepelne, akkor ResourceURI-nak a DMTF „All Classes" URI-ját kéne használni[10].

Ha elírjuk a szűrő dialektusát, akkor a következő hibaüzenetet kapjuk:

```
The requested filtering dialect is not supported.
```

Ha elírjuk a szűrőben lévő feltételt, akkor a következő hibát kapjuk (CQL szűrő esetén a hiba szövegét már a CIMOM adja vissza):

```
<s:Value>wsen:CannotProcessFilter</s:Value>
...
<s:Text xml:lang="en">syntax error in query.</s:Text>
```

7. Kapcsolatok lekérdezése

   A CIM-ben egy fontos fogalom a kapcsolóosztályok (association class), ennek a segítségével lehet osztályok közötti kapcsolatokat megvalósítani. A kapcsolatokat asszociációs osztályok jelzik, ezek mentén lehet navigálni a *wsmancli* ASSOCIATORS akciójának segítségével. Ez a háttérben egy Enumerate műveletre képződik le, ami az *Association* szűrődialektust használja.

   Mivel nem tudjuk még, hogy pontosan melyik osztályok példányait akarjuk lekérdezni, ezért ResourceURI-nak az úgynevezett összes osztály („All Classes") URI-t kell megadni:

```
# warning, this is wrong, a filter is required for associations
wsman -h localhost -u meres -p LaborImage associators
'http://schemas.dmtf.org/wbem/wscim/1/*'
```

   Erre a válasz a következő:

[10] Arra figyeljünk, hogy nincs SBLIM-specifikus „All Classes" URI, tehát a http://sblim.sf.net/wbem/wscim/1/* URI-ra InvalidResourceURI hibát dobna.

```
<s:Value>wsman:UnsupportedFeature</s:Value>
...
<s:Text xml:lang="en">The specified feature is not supported.</s:Text>
...
<wsman:FaultDetail>http://schemas.dmtf.org/wbem/wsman/1/wsman/faultDetail/
FilteringRequired</wsman:FaultDetail>
```

A `FaultDetail` rész segít ilyenkor, a kapcsolatok lekérdezésénél meg kell adni egy szűrőt is, ami legalább azt megmondja, hogy melyik példány kapcsolataira vagyunk kíváncsiak.

Egy példányt az osztályával és a kulcs attribútumaival tudunk azonosítani, ezeket kell most a szűrő szövegébe besűríteni:

```
wsman -h localhost -u meres -p LaborImage associators
'http://schemas.dmtf.org/wbem/wscim/1/*' --filter
'http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_ComputerSystem?→
Name="irfserver.irf.local",CreationClassName=Linux_ComputerSystem'
```

Itt most a `Linux_ComputerSystem` példányához bármilyen kapcsolaton keresztül is kapcsolódó példányokat kapjuk vissza.

Ha nem a kapcsolódó példányokat szeretnénk megkapni, hanem magukat a kapcsolatokat (azaz a kapcsolóosztályok példányait), akkor a fenti lekérdezésben `associators` helyett használjuk a `references` akciót.

Az *Associaton* szűrő dialektus eléggé sokrétű (lásd [2]), meg lehet például adni, hogy csak adott kapcsolóosztály mentén lévő kapcsolatokra vagyunk kíváncsiak:

```
wsman -R -h localhost -u meres -p LaborImage associators
'http://schemas.dmtf.org/wbem/wscim/1/*' --filter
'http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_ComputerSystem?
Name="irfserver.irf.local",CreationClassName=Linux_ComputerSystem,AssociationClas
sName=Linux_CSProcessor'
```

Érdemes még megnézni, hogy ilyenkor milyen WS-Management üzenet generálódik (az XML-t kicsit megvágtuk, hogy kiférjen):

```
<s:Header>
    <wsa:Action>http://schemas.xmlsoap.org/ws/2004/09/enumeration/Enumerate</wsa:Action>
    <wsa:To>http://localhost:5985/wsman</wsa:To>
    <wsman:ResourceURI>http://schemas.dmtf.org/wbem/wscim/1/*</wsman:ResourceURI>
    <wsa:MessageID>uuid:e971c54f-bcb8-1cb8-8002-8c3a19290c00</wsa:MessageID>
    <wsa:ReplyTo>
     <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
     </wsa:Address>
    </wsa:ReplyTo>
  </s:Header>
  <s:Body>
    <wsen:Enumerate>
      <wsman:Filter Dialect="http://schemas.dmtf.org/wbem/wsman/1/cimbinding/associationFilter">
        <wsmb:AssociatedInstances>
          <wsmb:Object>
            <wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
            </wsa:Address>
            <wsa:ReferenceParameters>
              <wsman:ResourceURI>
```

```
              http://sblim.sf.net/wbem/wscim/1/cim-schema/2/Linux_ComputerSystem
            </wsman:ResourceURI>
            <wsman:SelectorSet>
              <wsman:Selector Name="Name">irfserver.irf.local</wsman:Selector>
              <wsman:Selector Name="CreationClassName">Linux_ComputerSystem</wsman:Selector>
            </wsman:SelectorSet>
          </wsa:ReferenceParameters>
        </wsmb:Object>
        <wsmb:AssociationClassName>Linux_CSProcessor</wsmb:AssociationClassName>
      </wsmb:AssociatedInstances>
    </wsman:Filter>
  </wsen:Enumerate>
</s:Body>
```

Ha a lekérdezés során a következő hibát kapjuk

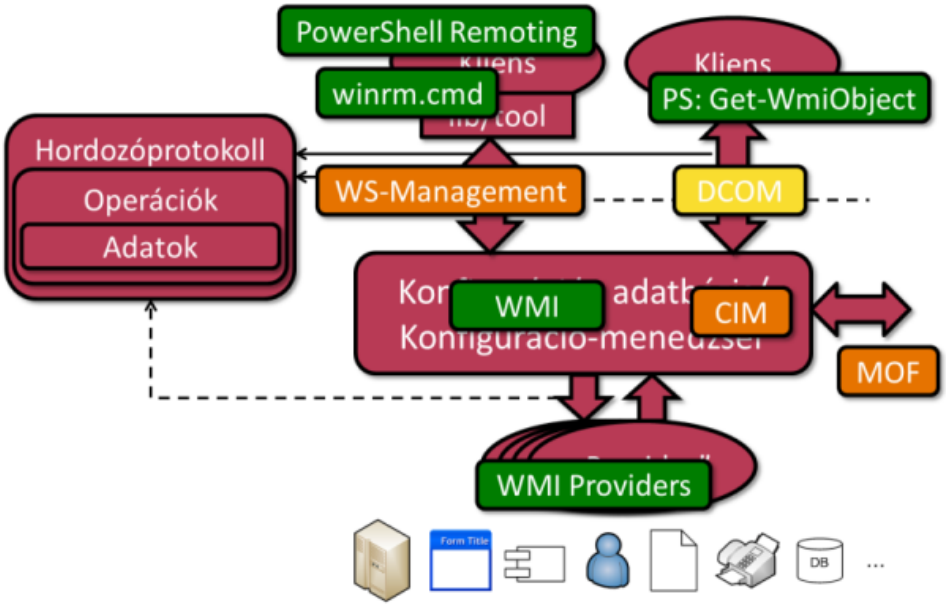
```
Provider not found or not loadable
```

akkor érdemes megnézni az *sfcbd* kimenetét is, mert valamelyik provider regisztrációja hibás (akár egy teljesen más asszociációs osztály hibája is okozhatja ezt, mert ilyenkor végignézi az összes a repository-ban lévő kapcsolóosztályt). Ilyenkor vagy megpróbáljuk megjavítani az adott providert, vagy, ha nincs rá épp szükség, akkor eltávolítjuk az sfcb repository-ból.

2 Windows: WMI, WinRM

A feladatokat egy Windows 7 virtuális gépen fogjuk végrehajtani. A Windows 7 már alapból tartalmazza a CIMOM-ot (WMI Object Manager), a WS-Management klienst és kiszolgálót (WinRM) és az ezek kezeléséhez szükséges PowerShell 2.0-s cmdleteket, így a feladatunk pusztán annyi lesz, hogy ezeket megfelelően beállítsuk.

Az általános konfigurációkezelési ábránkat tehát a következő módon fedik le az eszközök.



9. ábra: Konfigurációkezelési technológiák Windowsra

2.1 WMI használata

A következő feladatban egyszerű WMI lekérdezéseket fogunk végrehajtani helyi és távoli gépen.

1. PowerShell WMI Explorer

   Hogy könnyebben eligazodjunk, hogy milyen CIM osztályok és példányok vannak a rendszerben, érdemes a *PowerShell WMI Explorer WPF Edition*[11] kis segédprogramot letölteni. A letöltés után a fájl tulajdonságlapján az `Unblock` megnyomásával oldjuk fel a távoli letöltés miatti zárolást. A WMI Explorer szkript használatához a PowerShell konzolt *Single-Threaded Apartment* (STA) módban kell indítani a `-sta` paraméterrel:

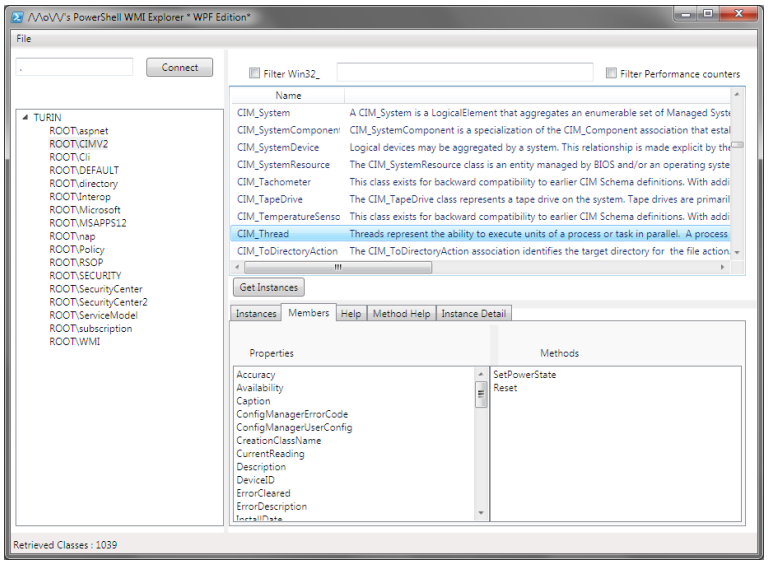
   ```
   powershell -sta
   ```

   Majd elindítani a letöltött szkriptet:

   ```
   .\WpfWmiExplorer.ps1
   ```

   A következőhöz hasonló kép fogad majd minket:

[11] http://thepowershellguy.com/blogs/posh/pages/powershell-wmi-explorer.aspx



10. ábra: PowerShell WMI Explorer WPF Edition

Itt ki lehet listázni az egyes névterekben szereplő osztályokat, azok részletes leírását, valamint le lehet kérdezni a példányaikat.

(Ennél egy picit több funkcióit valósít meg a `wbemtest.exe`, csak annak nehezebben használható a GUI-ja.)

2. WMI kezelésére szolgáló cmdletek

Keressük meg a WMI kezelésére szolgáló cmdleteket:

```
Get-Command -type Cmdlet | ? {$_.Name -like "*WMI*"}
```

Ezek közül mi legtöbbször a `Get-WmiObject` cmdletet fogjuk használni, de érdemes a többiről is tudni.

Nézzük néhány példát a Get-WmiObject használatára:

```
Get-Help Get-WmiObject -examples
```

Ezzel nagyjából képet lehet kapni, hogy mit tud a Get-WmiObject. Érdemes viszont rászánni az időt, hogy a teljes leírást is megnézzük (ezt egyszer úgyis meg kell tenni), különben folyamatosan elakadunk majd csak később.

```
Get-Help Get-WmiObject -full | more
```

Ezzel a tudással felvértezve most már le is tudunk valamit kérdezni.

```
Get-WmiObject CIM_Memory
```

Válaszként `System.Management.ManagementObject` típusú objektumokat kapunk vissza, amiknél szépen tulajdonságokkal érjük el az egyes attribútumokat. Ez alapján már könnyen tudunk szűrni vagy rendezni PowerShellben a korábban tanult módon.

```
Get-WmiObject CIM_Memory | select Name, Status, InstalledSize | where
{$_.InstalledSize -gt 1024}
```

- Keressünk ki további 2 CIM osztályt, és kérdezzük le azok példányait.
- Szűrjük az így visszakapott listát, majd számoljuk ki a maximumát az egyik tulajdonságnak.

3. WMI lekérdezés távolról

   Ha egy távoli gépről akarunk információt lekérni, akkor csak a -ComputerName paramétert kell megadni:

   ```
   Get-WmiObject CIM_Memory -ComputerName 192.168.21.151
   ```

   Erre alapesetben a következő hibaüzenet a válasz:

   ```
   The RPC server is unavailable. (Exception from HRESULT: 0x800706BA)
   ```

   Ahogy az előadáson is szerepelt, a WMI távoli lekérdezésekhez DCOM-ot használ, és az ehhez szükséges portok nincsenek kinyitva alapértelmezetten. A fólián megadott paranccsal nyissuk ki a távoli gépen a megfelelő portot:

   ```
   netsh firewall set service RemoteAdmin enable
   ```

   Most megismételve a távoli lekérdezést már más hibát kapunk:

   ```
   Access is denied. (Exception from HRESULT: 0x80070005 (E_ACCESSDENIED))
   ```

   A Windows ilyenkor a helyi felhasználónk adataival próbált áthitelesíteni. Ha ez nem létezik a távoli gépen, akkor ilyen hibát kaphatunk. A `-Credential` paraméterrel tudjuk megadni, hogy milyen felhasználó nevében csatlakozzon. Ilyenkor egy dialógusablakban vagy a konzolon a parancs futtatásakor bekéri hozzá a jelszót is.

   Az eredmény továbbra is *Access Denied*, holott olyan felhasználót adtunk meg, aki tagja a távoli gépen az *Administrators* csoportnak. Ez azért van, mert Vista óta a *User Account Control (UAC)* funkció miatt az ilyen felhasználók távoli hozzáférés esetén olyan biztonsági tokent kapnak, amiben nincsenek benne a rendszergazdai jogok. Bővebben lásd a megfelelő KB cikket [6]. A következő registry kulcs értékét kell 1-re állítani:

   ```
   HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\
   LocalAccountTokenFilterPolicy
   ```

   Így már sikeresen le lehet távoli gépek adatait is kérdezni.

4. Szűrés távoli lekérdezés esetén

   Arra figyeljünk oda, hogy távoli lekérdezés esetén csak a szükséges adatokat kérdezzük le, és inkább a távoli gépen szűrjünk. Erre valók a WQL (WMI Query Language) lekérdezések. A részleteket lásd az előadás fóliákon, itt most csak egy egyszerű példát nézünk:

```
Get-WmiObject -ComputerName 192.168.21.151 -Credential meres -Query "Select
name, state FROM Win32_Service WHERE StartMode = 'Auto'"
```

WQL segítségével bonyolultabb lekérdezéseket is meg tudunk fogalmazni.

5. Kapcsolódó példányok lekérdezése

   Még egy érdekes feladat van hátra, mégpedig az, amikor szeretnénk egy adott példányhoz kapcsolódó másik példányokat lekérdezni.

   A CIM-ben erre a kapcsolóosztályok (association class) szolgálnak. Például a *Win32_DiskDriveToDiskPartition* a *Win32_DiskDrive* és a *Win32_DiskPartition* osztályokat köti össze, a *Win32_SoftwareFeatureSoftwareElements* pedig egy *Win32_SoftwareFeature* példányhoz tartozó *Win32_SoftwareElement* példányokat adja meg.

   Ilyen osztályok mentén navigálhatunk kézzel is, de egyszerűbb az `ASSOCIATORS OF` kulcsszó használata [8]. Például így kérdezhetjük le egy konkrét *Win32_LogicalDisk* példányhoz tartozó összes kapcsolódó példányt.

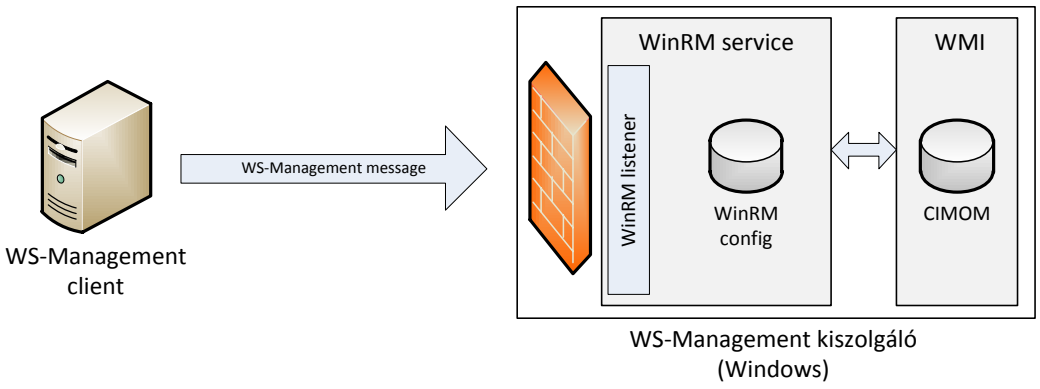
```
Get-WmiObject -Query "ASSOCIATORS OF {Win32_LogicalDisk.DeviceID='C:'}"
```

   Ez eredményként az összes kapcsolóosztályt megvizsgálja, így kapunk például *Win32_ComputerSystem*, *Win32_DiskPartition* és *Win32_Directory* példányt is. Ha csak egy konkrét kapcsolatra vagyunk kíváncsiak, akkor az `AssocClass` szűrőfeltételt lehet használni.

   Érdemes megnézni az ASSOCIATORS OF leírását, még tud további hasznos dolgokat.

2.2 WinRM használata

A WinRM a Microsoft WS-Management protokollt használó távoli menedzsment implementációja.



11. ábra: A WinRM architektúrája

A WinRM maga egy szolgáltatás, ami WS-Management üzeneteket fogad (vagy küld), és az abban megfogalmazott kéréseket végrehajtja a megadott erőforráson, pl. egy WMI példányon. A szolgáltatás alapesetben a megvalósítást adó kódból, egy saját belső

konfigurációs adatbázisból és egy vagy több úgynevezett *figyelőből* (listener) áll, amik megadják, hogy melyik helyi IP-címen, milyen porton és milyen csatornán (HTTP, HTTPS) figyeljen a WinRM szolgáltatás. A WS-Management kérések fogadásához a megadott porton természetesen a tűzfalon is át kell engedni.

1. Ismerkedés a WinRM-mel

   Gyors áttekintést ad a WinRM-ről, valamint arról, hogy hogyan tudjuk PowerShellből kezelni a következő súgó téma:

   ```
   Get-Help about_wsman | more
   ```

   Ezt érdemes elolvasni, sokat segít az orientálásban.

   Az aktuális telepített PowerShell és WinRM verziót így tudjuk ellenőrizni:

   ```
   $PSversiontable
   ```

   A WinRM engedélyezését végzi el a `Set-WSManQuickConfig` cmdlet. A leírásában benne van, hogy mit csinál:

   ```
   The cmdlet performs the following:
   1. Checks whether the WinRM service is running. If the WinRM service is not
   running, the service is started.
   2. Sets the WinRM service startup type to automatic.
   3. Creates a listener to accept requests on any IP address. By default, the
   transport is HTTP.
   4. Enables a firewall exception for WinRM traffic .
   ```

   A végrehajtásához *rendszergazdaként* kell elindítani a PowerShell konzolt.

   Ezek a beállítások csak ahhoz kellenek, ha a WinRM-es gépünket kiszolgálóként akarjuk használni, tehát róla akarunk adatokat lekérdezni. Ha a WinRM-es gép a kommunikációban kliensként viselkedik, tehát ő kérdez le adatokat, akkor elég csak annyi, hogy a WinRM szolgáltatást elindítjuk kézzel.

   Meg lehetne adni neki egy `-UseSSL` kapcsolót is, ilyenkor nem a WS-Management protokollhoz rendelt HTTP porton (5985) hanem a HTTPS porton (5986) figyelne. Ehhez viszont kell a gépre telepíteni egy tanúsítványt, ami a számítógép nevére szól (ez lehet akár nem hiteles tanúsítvány is, amit pl. az *OpenSSL* program segítségével generáltunk magunknak).

   A parancs végrehajtásakor kaphatjuk a következő hibaüzenetet:

   ```
   WinRM firewall exception will not work since one of the network connection types
   on this machine is set to Public. Change the network connection type to either
   Domain or Private and try again.
   ```

   Ehhez át kell állítani a *Network and Sharing Center* ablakban a hálózati hely típusát *Work*re (figyelem, ennek például az is lehet a következménye, hogy megnyitja a fájlmegosztásokhoz tartozó portot). Vagy ha ez nem lehetséges (mert például olyan VMware virtuális interfész van a gépen, amit nem lehet a Public profilról átállítani)

vagy nem akarjuk, akkor a Set-WSManQuickConfig által elvégzett lépéseket kézzel is megcsinálhatjuk:

```
Get-Service winrm | Start-Service
Get-Service winrm | Set-Service -StartupType Automatic
New-ItemProperty `
 –path HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System `
 –name LocalAccountTokenFilterPolicy –propertyType DWord –value 1
New-WSManInstance winrm/config/Listener `
 -SelectorSet @{Transport="HTTP";Address="*"}
netsh advfirewall firewall set rule name="Windows Remote Management (HTTP-In)" →
 new enable=yes
```

Ha ellenőrizni akarjuk, hogy sikerül-e távolról csatlakozni, akkor használjuk a *Test-WSMan* cmdletet. Ez a WS-Management IDENTIFY műveletét használja.

```
Test-WSMan -ComputerName 192.168.21.151
```

Ilyenkor névtelen módon csatlakozik a távoli géphez.

A tipikus csatlakozási problémák megoldását sorolja fel a következő súgó oldal:

```
Get-Help about_Remote_Troubleshooting
```

2. Egyszerű lekérdezés WinRM segítségével

Vegyük elő valamelyik szokásos CIM osztályunkat, és kérdezzük le a példányait WS-Management segítségével. Erre jó a `Get-WSManInstance` cmdlet. Ennek is egy erőforrás URI-t kell megadni. Szerencsére itt használhatunk aliasokat, nem kell a teljes prefixet mindig kiírni. Az aliasok listája itt található:

```
winrm help alias
```

Nézzünk akkor ez alapján egy egyszerű Enumerate kérést.

```
Get-WSManInstance wmicimv2/Win32_Processor -ComputerName 192.168.21.151 `
 -Enumerate
```

Erre egy nagyon tipikus választ fogunk kapni:

```
The WinRM client cannot process the request. If the authentication scheme is
different from Kerberos, or if the client computer is not joined to a domain,
then HTTPS transport must be used or the destination machine must be added to the
TrustedHosts configuration setting. Use winrm.cmd to configure TrustedHosts. Note
that computers in the TrustedHosts list might not be authenticated. You can get
more information about that by running the following command: winrm help config.
```

Olvassuk végig a hibaüzenetet, mert pontosan leírja, hogy mi a gond, és mi a megoldás. Mivel nem tartományi környezetben vagyunk, így nem lehet Kerberost használni. Ekkor ha nem akarunk SSL-t használni, akkor a távoli gépet fel kell venni a helyi gép TrustedHosts listájába, és ezzel vállalva, hogy küldhetünk neki nem titkosított tartalmat is.

A WinRM beállításait a winrm.cmd parancssori eszközzel módosíthatjuk, de használhatjuk erre a WSMAN: PowerShell providert is[12]. Nyissunk egy új PowerShell konzolt *rendszergazdaként*, majd hajtsuk végre a következetőket:

```
cd WSMAN:
cd localhost
ls
```

Figyelem: ehhez a rendszergazda jogú felhasználónknak kell jelszóval rendelkeznie, ha üres a jelszava, akkor *Access Denied* hibát kapunk.

Itt találhatók a WinRM beállításai: kliens, szolgáltatás, és a bejövő kéréseket fogadó úgynevezett listenerek (ezek döntik el, hogy melyik IP-n és milyen porton figyel a WinRM). Egy friss Windows 7 gépen a következőt kapjuk:

```
   WSManConfig: Microsoft.WSMan.Management\WSMan::localhost

Name                    Value
----                    -----
MaxEnvelopeSizekb       150
MaxTimeoutms            60000
MaxBatchItems           32000
MaxProviderRequests     4294967295
Client
Service
Shell
Listener
Plugin
ClientCertificate
```

Most nekünk a helyi gép kliens beállításait kell módosítani, így váltsunk át arra.

```
cd Client
ls
```

- Nézzük végig, hogy milyen főbb beállítások vannak!
- Milyen hitelesítési módszerek vannak jelenleg a kliensben engedélyezve?

Állítsuk át a `TrustedHosts` értékét, adjuk hozzá a távoli gép IP-címét is.

```
Set-Item .\TrustedHosts -Value "192.168.21.151"
```

(Arra figyeljünk, hogy ha volt már valami korábban a TrustedHosts mezőben, akkor annak az értékét tartsuk meg, ha kell még. Az egyes elemek vesszővel vannak elválasztva a TrustedHosts értékében.)

Ha ezzel megvagyunk, akkor elvileg már megy a lekérdezés.

[12] Harmadik lehetőség, hogy csoportházirendet használunk erre. Ezt akár nem tartományi gépen is megtehetjük. Nyissunk meg egy MMC-t, adjuk hozzá a *Group Policy Object Editor* snap-in konzolt a helyi gépen. A beállítások a *Computer Configuration / Administrative Templates / Windows Components / Windows Remote Management* rész alatt találhatóak.

```
Get-WSManInstance wmicimv2/Win32_Processor -ComputerName 192.168.21.151
 -Credential meres -Enumerate
```

3. Konkrét példány lekérdezése

   Ha nem felsorolni akarjuk egy adott osztály összes példányát, hanem egy konkrét példányt akarunk lekérdezni, akkor *Selector*okban meg kell adni az összes kulcs attribútumának az értékét. A kulcsok kitalálásához segítségként egy WMI lekérdezéssel megnézhetjük az adott példány adatait:

   ```
   Get-WmiObject Win32_NetworkAdapter | select __PATH
   ```

   Itt szerepel az egyes példányok teljes neve. A kulcs attribútumokat egy hash-táblában kell összegyűjteni és átadni a `SelectorSet` paraméternek:

   ```
   Get-WSManInstance -ComputerName 192.168.21.151 -Credential meres -ResourceURI
    wmicimv2/Win32_NetworkAdapter -SelectorSet @{DeviceID=1}
   ```

4. Szűrés távoli lekérdezésben

   Többféle módon lehet szűrni. Ha egy osztály példányai közül akarjuk azok egy részhalmazát kiszűrni, akkor használhatunk *Selector* dialektusú szűrőt.

   ```
   Get-WSManInstance -ComputerName 192.168.21.151 -Credential meres -ResourceURI
   wmicimv2/Win32_NetworkAdapter -Enumerate -Dialect Selector -Filter
   'AdapterType="Tunnel";Speed="100000"'
   ```

   (Arra figyeljünk, hogy itt az egyes attribútumokat pontosvesszővel kell elválasztani és nem vesszővel, mint a wsmancli esetében.[13])

   Ha bonyolultabb lekérdezéseket akarunk végrehajtani, és a távoli gép Windowst futtat, akkor használhatunk WQL lekérdezéseket. Ilyenkor mivel nem egy adott osztályt címzünk meg, ezért a ResourceURI-ban az windowsos „All Classes” URI-t kell használni:

   ```
   Get-WSManInstance -ComputerName 192.168.21.151 -Credential meres -ResourceURI
   wmicimv2/* -Enumerate -Dialect WQL -Filter "SELECT name, index FROM
   Win32_NetworkAdapter WHERE AdapterType = 'Tunnel' AND Index > 11"
   ```

5. Töredék (fragment) kezelés

   Ha a visszaadott válaszból csak egy konkrét értékre vagyunk kíváncsiak, akkor kérhetjük, hogy csak azt adja vissza a kiszolgáló. Erre a `-Fragment` paraméter való.

   A paraméter pontos jelentését a WS-Man szabvány leírásában találhatjuk (7.7 Fragment-Level Access) [1]. Eszerint a visszaadandó XML dokumentumhoz egy XPath lekérdezést adhatunk meg, és így csak az általa kijelölt XML csomópontokat kapjuk majd vissza. A szabvány függeléke tartalmazza, hogy az XPath milyen részhalmaza támogatott, ám ez elég szűk (a WinRM a *Level 1* szintet implementálta).

[13] A szabvány ezt nem definiálja, mert a szabványban csak a szűrő XML reprezentációja van definiálva.

Valami hasonló szerkezetű XML dokumentumot kapunk általában vissza (ezt Wiresharkból tudjuk például megnézni, vagy ha a lekérdezést elmentjük egy változóba, és annak a `psbase` tulajdonságát kérdezzük le):

```
<wsman:Results xmlns:wsman="http://schemas.dmtf.org/wbem/wsman/1/wsman/results">

<p:Win32_Processor xsi:type="p:Win32_Processor_Type" xml:lang="en-US" xmlns:xsi=
"http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://schemas.microsoft.co
m/wbem/wsman/1/wmi/root/cimv2/Win32_Processor" xmlns:cim="http://schemas.dmtf.or
g/wbem/wscim/1/common">
    <p:AddressWidth>32</p:AddressWidth>
    <p:Architecture>9</p:Architecture>
    <p:Availability>3</p:Availability>
...
```

A támogatott XPath kifejezések alapján (pl. nincsen „a | b” operátor) ez praktikusan azt jelenti, hogy egy-egy csomópontot tudunk kiválasztani a `-Fragment` segítségével.

Ráadásul `-Enumerate` esetén nem lehet a WinRM kliensben töredéket megadni, különben a következő hibát kapjuk:

```
Parameter set cannot be resolved using the specified named parameters.
```

6. Kapcsolóosztályok használata

Zárásként nézzük meg itt is, hogy hogyan lehet a kapcsolóosztályok mentén adatokat lekérdezni.

Két Windows esetén „csalhatunk”, és a lekérdezés dialektusának megadhatunk WQL lekérdezéseket, ilyenkor például működik az ASSOCIATORS OF kulcsszó.

Azonban ha tényleg platform-független módon szeretnénk kezelni a kapcsolatokat, akkor a WS-Management *Association* típusú szűrőjét kell használni. A témához kapcsolódó hivatalos WinRM dokumentáció [9] viszonylag szűkszavú, a Get-WSManInstance dokumentációja pedig hibás (például egyes verziókban még szerepel a már aktuálisan nem létező -References kapcsoló). Egy jó összefoglaló található itt [10] a témáról.

A lekérdezések pontos jelentését a DMTF szabványban találhatjuk meg (8.2 Association Queries) [2]. Nézzünk itt most egy konkrét példányt, hogy WinRM-ből hogyan lehet ezt használni.

```
Get-WSManInstance -ComputerName 192.168.21.151 -ResourceURI wmicimv2/* `
 -Dialect Association -Filter "{object=Win32_LogicalDisk?deviceid=C:}" `
 -Enumerate -Credential meres -Associations
```

Figyelem: kapcsolatok lekérdezésekor a kliensen a PowerShellt is rendszergazdai jogokkal kell futtatni, különben hozzáférési hibát kapunk.

Ennek eredményeképp a kapcsolatok példányait kapjuk meg (a `-Associations` kapcsoló miatt), és nem a kapcsolódó példányokat. Ilyenkor hasonló kimenetet láthatunk:

```
type           : p:Win32_LogicalDiskRootDirectory_Type
GroupComponent : GroupComponent
PartComponent  : PartComponent

type            : p:Win32_LogicalDiskToPartition_Type
Antecedent      : Antecedent
Dependent       : Dependent
```

A GroupComponent és az Antecedent összetett objektumok, azért nem írja ki a konkrét értékét. De a konkrét visszakapott objektumon a következő tulajdonságok mentén elérjük:

```
.GroupComponent.ReferenceParameters.SelectorSet.Selector
```

A másik lehetőség, hogy a konkrét példányokat kérjük vissza:

```
Get-WSManInstance -ComputerName 192.168.21.151 -ResourceURI wmicimv2/* -dialect
association -filter
"{Object=Win32_DiskDrive?deviceID=\\\\.\\PHYSICALDRIVE0;AssociationClassname=Win3
2_DiskDriveToDiskPartition}" -enumerate -credential meres
```

Tipikus hibaüzenet szokott lenni a következő:

```
Get-WSManInstance : The data source could not process the filter. The filter
might be missing, invalid or too complex to process. If a service only supports a
subset of a filter dialect (such as XPath level 1), it may return this fault for
valid filter expressions outside of the supported subset. Change the filter and
try the request again.
```

Ezt akkor kaphatjuk, ha például a kapcsolóosztályok lekérése során is megpróbáljuk megadni az AssociationClassName szűrőt. Figyeljük meg a két eset leírásában [10], hogy különböző elemeket használhatunk a szűrőben.

3 Platformok közötti lekérdezések

Az igazi erejét az adja ezeknek a szabványoknak és technológiáknak, hogy a lekérdezések platformok között is működnek. Nézzünk erre egy példát.

3.1 WS-Management Linux klienssel és Windows szolgáltatással

Nézzük most azt az esetet, amikor kliensnek a wsmancli programot használjuk Linuxról, a kiszolgáló pedig egy WinRM-et használó Windows lesz.

1. Kapcsolódás kipróbálása

```
wsman identify -h 192.168.21.151 -P 5985
```

Erre *authentication failed* választ kapunk. Ki kéne akkor választani, hogy milyen hitelesítési módszert használjunk. A wsman tud HTTP Basic-et és GSS-t (amivel Kerberost lehet használni).

A WinRM kiszolgáló beállításait így nézhetjük meg (rendszergazdai PowerShell konzolból):

```
cd wsman:
cd .\localhost\Service\Auth
ls
```

Eredmény:

```
Name                    Value
----                    -----
Basic                   false
Kerberos                true
Negotiate               true
Certificate             false
CredSSP                 false
```

Kerberost ugyan mindkettő tud, ahhoz azonban kéne egy Kerberos szerver (például egy Active Directory tartományvezérlő). Ha ez nincs, akkor marad a Basic. Ez nem titkosítva küldi a jelszót, de legalább biztos mindenki ismeri. Ehhez engedélyezni kell a Basic hitelesítési módszert a WinRM kiszolgálón is:

```
Set-Item WSMan:\localhost\Service\Auth\Basic -Value true
```

Egy dolog kell még a WinRM oldalán. Alapból nincs engedélyezve, hogy titkosítatlan csatornán kommunikálhat bárkivel is, ezért ezt a hibaüzenetet kapjuk:

```
Connecting to remote server failed with the following error message : The WinRM
client cannot process the request. Unencrypted traffic is currently disabled in
the client configuration. Change the client configurati on and try the request
again.
```

Így ezt most be kell kapcsolni:

```
Set-Item WSMan:\localhost\Service\AllowUnencrypted -value true
```

Vissza a wsman klienshez, és most adjuk meg, hogy Basic hitelesítést akarunk használni, és mi a felhasználó és jelszó:

```
wsman identify -h 192.168.21.151 -P 5985 --auth basic -u meres -p LaborImage
```

Most már megy tökéletesen az IDENTITY művelet.

Ha gondunk lenne, első lépésként érdemes Wiresharkban megnézni a forgalmat, és például ellenőrizni, hogy jól adja-e át a Base64 segítségével kódolt felhasználót és jelszót, lefolyik-e rendesen a HTTP Basic hitelesítés stb. Például ha a Windows gép a kliens, akkor előfordul néha az a probléma, hogy a felhasználónév elé egy \ jelet rak, és ez csak a hálózati forgalomban látszik[14].

2. Adatok lekérdezése

Nézzünk akkor kezdésnek egy ENUMERATE műveletet:

```
wsman enumerate
'http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/CIM_Processor' -h
192.168.21.151 -P 5985 --auth basic -u meres -p LaborImage
```

Egy dologra kell figyelni. Az erőforrás URI-ban itt a Microsoft specifikus prefixet használtuk. A DMTF által szabványosított prefix használata nem megy tökéletesen Windows 8 előtt [7]. Ha a távoli gép Windows 8, akkor már megy szépen a következő lekérdezés is (arra kell figyelni, hogy meg kell adni a névteret is):

```
wsman -h 192.168.160.131 --auth basic -u meres -p LaborImage enumerate
http://schemas.dmtf.org/wbem/wscim/1/cim-
schema/2/CIM_Processor?__cimnamespace=root/cimv2
```

Nézzünk meg egy GET műveletet is:

```
wsman get
'http://schemas.microsoft.com/wbem/wsman/1/wmi/root/cimv2/Win32_Processor?DeviceID=CPU0'
-h 192.168.21.151 -P 5985 --auth basic -u meres -p LaborImage
```

Ez a része is működik, így különböző platformok között is tudunk adatokat lekérdezni. Bár elsőre egy XML fájl visszakapása nem tűnik nagy haditettnek, de ha belegondolunk ez egy egész sor érdekes alkalmazás előtt nyitja meg az utat.

3. Biztonsági beállítások

Az első részben beállított beállításokkal megy a rendszer, csak éppen semmi védelem nincsen benne. A jelszavak és a teljes forgalom nyílt szövegként utazik, egyik fél sem ellenőrzi a másik kilétét.

[14] Ilyenkor megoldás lehet például, hogy a Get-Credential cmdletet átállítjuk, hogy konzolon kérje be a jelszót: http://social.technet.microsoft.com/Forums/pl-PL/winserverpowershell/thread/8eb1a046-acbf-4fda-b4b3-e7599dcf53a3

A következő lépés az lenne, hogy ezeket beállítjuk valami elfogadható szintre. Több lehetőségünk is van.

- Kerberos használata: akár AD, akár egy linuxos Kerberos kiszolgálóval.
- SSL használata: ez a teljes forgalmat titkosítaná. Ehhez a windowsos gépen kell egy tanúsítványt telepíteni, majd átállítani a listenert.

3.2 WS-Management Windows klienssel és Linux kiszolgálóval

Nézzük most meg a másik esetet, tehát a WinRM lesz a kliens, és egy openwsmand a kiszolgáló. Az előző fejezetben elmondottak továbbra is érvényesek, a Basic lesz a közösen ismert hitelesítési protokoll, és egyelőre sima HTTP-n próbálkozunk.

1. Biztonsági beállítások a kliensen

   Első lépésben engedélyezzük a Basic hitelesítést a WinRM kliens beállításaiban.

   ```
   Set-Item WSMan:\localhost\Client\Auth\Basic -Value true
   ```

   Ha csatlakozni próbálnánk, akkor a TrustedHosts beállításra panaszkodik, így állítsuk be azt a 2.2 fejezetben megismert módon:

   ```
   cd wsman:
   cd localhost\client
   Set-Item .\TrustedHosts -Value "192.168.21.151"
   ```

   (Az IP-címre természetesen helyettesítsük be a kiszolgáló IP-címét.)

2. Kiszolgáló beállítása

   A kiszolgáló oldalon ellenőrizzük, hogy fut-e a CIM szerver és az openwsmand kiszolgáló, be van-e állítva a Basic hitelesítés a konfigurációs állományában, végül pedig, hogy helyben tudunk-e lekérdezni róla.

3. Kapcsolat ellenőrzése

   Ezután már csak ellenőrizni kell a kapcsolatot.

   ```
   Test-WSMan -ComputerName 192.168.21.150 -Authentication basic -Credential meres
   ```

   Ha sikeres, akkor próbáljunk lekérdezni valamit:

   ```
   Get-WSManInstance -ComputerName 192.168.21.150 -Authentication basic -Credential
   meres -ResourceURI cimv2/CIM_Processor -Enumerate
   ```

   Arra figyeljünk, hogy itt most a platform-független (DMTF) vagy a Linux-specifikus URL prefixeket és osztályneveket kell használni.

4 Összefoglalás

A gyakorlat során megnéztünk különböző konfigurációkezelési technológiákat a gyakorlatban. A célunk az volt, hogy különböző platformról tudjunk változatos konfigurációs adatokat lekérdezni és ezeket az alkalmazásainkban felhasználni.

Az általános módszer és szabványkészlet a következő volt. A CIM definiált egy általános adatmodellt, ilyen modelleket tárolnak a CIMOM-jaink. A CIMOM-ot többféle protokollon lehet elérni, a gyakorlat során a CIM-XML és a WS-Management protokollokat néztük meg.

Linux platformon megismerkedünk az sfcb CIMOM-mal, és a CIM-XML protokollt használó wbemcli lekérdező eszközzel. A CIMOM-unk számára egy WS-Management interfészt biztosított pluszban az openwsman, ezt a wsmancli eszközzel tudtuk lekérdezni.

Windows esetén a WMI biztosította a CIMOM-ot. Ezt vagy a nem szabványos DCOM felületen lehet elérni, vagy pedig a WinRM által biztosított WS-Management felületen. Mindkét esetben PowerShell cmdleteket használtunk a lekérdezésekhez.

5 További információ

Általános szabványok

[1] DMTF. „Web Services for Management (WS Management)”, 1.1.0, DSP0226, 2010. URL: http://dmtf.org/sites/default/files/standards/documents/DSP0226_1.1.pdf

[2] DMTF. „WS-Management CIM Binding Specification”, 1.1.0, DSP0227. URL: http://www.dmtf.org/sites/default/files/standards/documents/DSP0227_1.1.0.pdf

[3] DMTF. „CIM Query Language Specification”, 1.0.0, DSP0202. URL: http://dmtf.org/sites/default/files/standards/documents/DSP0202_1.0.0.pdf

Linux

[4] Praveen Kumar Paladugu. „WBEM Based Management in Linux”, Dell Technical White Paper, http://en.community.dell.com/dell-blogs/enterprise/b/tech-center/archive/2011/02/28/wbem-based-management-in-linux.aspx

[5] RMS’s GDB Debugger Tutorial, URL: http://www.unknownroad.com/rtfm/gdbtut/

Windows

[6] Microsoft, Description of User Account Control and remote restrictions in Windows Vista, Knowledge Base article 951016, http://support.microsoft.com/kb/951016

[7] Micskei Zoltán, „CIM osztályok lekérdezése WinRM-ben DMTF URI-val”, http://blog.inf.mit.bme.hu/?p=144

[8] MSDN. „ASSOCIATORS OF Statement”,http://msdn.microsoft.com/en-us/library/aa384793%28v=vs.85%29.aspx

[9] MSDN. „DMTF Profile Discovery Through Association Traversal”, http://msdn.microsoft.com/en-us/library/ee309363%28VS.85%29.aspx

[10] WMI Blog. „Association Traversal Using WSMan cmdlets”, http://blogs.msdn.com/b/wmi/archive/2009/05/02/association-traversal-using-wsman-cmdlets.aspx

6 Függelék

A függelékbe kiegészítő, régebbi anyagok kerültek be, amik esetleg hasznosak lehetnek az érdeklődőknek.

6.1 wsmancli 2.2.5 segmentation fault hiba

A wsmancli 2.2.5-ös verziójában egy egyszerű ENUMERATE kérés esetén is *segmentation fault* hibával leállt a program. Tanulságos lehet, hogy hogyan lehet megkeresni a hiba okát, és azt egyszerűen megoldani.

```
wsman enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage
```

A lekérdezés eredménye a wsman 2.2.5-ös verziójával:

```
<s:Envelope xmlns:s=http://www.w3.org/2003/05/soap-envelope
...
  <wsen:EnumerateResponse>
      <wsen:EnumerationContext>c229af53-9fa3-1fa3-8002</wsen:EnumerationContext>
  </wsen:EnumerateResponse>
..
</s:Envelope>
Segmentation fault
```

Visszakapjuk tehát a WS-Management protokoll üzenetét egy SOAP[15] borítékban, azonban a feldolgozás közben *segmentation fault*ot kapunk. Nem túl szép.

Mit lehet ilyenkor tenni? Nyilván az egyik lehetőség, hogy feladjuk, és elkezdünk panaszkodni. Ettől azonban a probléma még nem fog megoldódni. Egy fokkal jobb megoldás, ha rákeresünk, hogy találkozott-e más ezzel a problémával. Sajnos túl sok találat nincs a releváns kereső kifejezésekre (pl. „wsman segmentation fault enumeration"). Akkor most mit tegyünk? Ha a Google se tudja a választ, akkor hogyan tovább?

Első körben jó lenne tudni, hogy pontosan hol akad el a program. Ezt egy debuggerrel könnyen meg tudjuk nézni. Linux alatt a gdb az egyik gyakran használt parancssori debugger. Ehhez elég sok gyorstalpalót lehet találni, pl. [5]. Nekünk most pusztán annyi kell, hogy elindítsuk, és a hibánál nézzük meg az aktuális verem állapotot.

```
gdb wsman
```

Majd a gdb konzolján el kell indítani a programot a megfelelő argumentumokkal:

```
run enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage
```

Az eredmény:

```
Program received signal SIGSEGV, Segmentation fault.
0x004f6333 in strlen () from /lib/libc.so.6
```

[15] http://en.wikipedia.org/wiki/SOAP

Érdemes megnézni a stack trace-t:

```
(gdb) backtrace
#0  0x004f6333 in strlen () from /lib/libc.so.6
#1  0x0804ba4d in main ()
```

Sajnos mivel nincsenek debug szimbólumok a programhoz, ezért például sorszámokat és lokális változókat nem látunk. Erre figyelmeztet is a gdb az elején:

```
Reading symbols from /usr/bin/wsman...(no debugging symbols found)...done.
```

Annyit mindenesetre látunk, hogy az `strlen()` függvényt hívjuk meg nem megfelelő argumentumokkal.

Hogy könnyen tudjunk továbbhaladni, érdemes készíteni egy olyan verziót az wsman programból, amiben vannak debug szimbólumok is. Mivel elérhető a forrása, ezért ez, ha nem is triviálisan, de megoldható. A pontos részleteket itt most átugorjuk, a lényeg annyi, hogy a *configure* szkript futtatása előtt a C fordítónak meg kell adni, hogy generáljon debug szimbólumokat is (*export CFLAGS=-g*). Az így kapott verzióval már a következőket látjuk a debuggerben:

```
#1  0x0804ba4d in main (argc=Cannot access memory at address 0x0
) at wsman.c:447
447         int count = strlen(output_file) + 16;
```

A gond tehát a 447. sorban van, ahol az `output_file` mutatóra hívjuk meg az `strlen` függvényt. Mivel nem adtunk meg olyan bemeneti kapcsolót, ami kimeneti fájlt írna elő, ezért az output_file értéke NULL, így nyilván segmentation fault lesz az eredmény. Ha megnézzük a wsman.c forrásfájlt, akkor a WSMAN_ACTION_ENUMERATION ág lekezelésekor feltétel nélkül meghívjuk a `wsman_output_pull` függvényt, ami fájlba szeretne írni.

Az enumeration műveletnél tehát adjunk meg kimeneti fájlt:

```
wsman enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor'
-h localhost -P 5985 -u pegasus -p LaborImage -O ~/wsman.out
```

Ez már hiba nélkül lefut, és létrejön egy wsman.out nevű fájl, amibe az enumeration context lekérése kerül be, valamint egy wsman.out-1.xml fájl, amiben a lekért CIM objektumok vannak XML-be ágyazva.

Bár nem volt egyszerű, de azért sikerült adatokat lekérnünk a CIMOM-tól WS-Management protokollon keresztül.

A tanulság az, hogy az openwsman ugyan eléggé aluldokumentált és vannak benne hibák, de ha kicsit gondolkozunk és használjuk az eddig megszerzett tudásunkat, akkor meg lehet oldani az előkerülő hibákat.

6.2 OpenPegasus

A korábbi években az *OpenPegasus* nevű *CIM Object Manager* eszközt használtuk, ez a leírás áttekinti a beállításait és bemutat pár egyszerű lekérdezést.

1. OpenPegasus beállításai

   A fájlok a `/etc/Pegasus` könyvtárban vannak. (A *pem* kiterjesztésű fájlok digitális tanúsítványok.)

   - Milyen állományokat találunk itt, ezek mit tárolnak?

   Nézzük meg, hogy kinek van joga hozzáférni a CIM szerverhez!

   ```
   nano /etc/Pegasus/access.conf
   ```

2. OpenPegasus elindítása

   Indítsuk el a CIM kiszolgálót:

   ```
   sudo /etc/init.d/tog-pegasus start
   ```

   Kérdezzük le, hogy tényleg elindult-e:

   ```
   sudo /etc/init.d/tog-pegasus status
   ```

   Ilyenkor visszaadja az elindított cimserver folyamat azonosítóját (PID).

3. OpenPegasus futási beállításai

   Miután fut az OpenPegasus, megnézhetjük a futási idejű beállításait is:

   ```
   cimconfig -l -c
   ```

   - Engedélyezve van-e az SSL?

   Bizonyos paramétereket nem lehet futás közben módosítani, ilyenkor a módosításnál meg kell adni a –p kapcsolót, aminek a hatására ez csak a következő induláskor jut érvényre. Példa:

   ```
   sudo cimconfig -s enableHttpConnection=true -p
   ```

4. Providerek listázása

   Nézzük meg, hogy milyen providerek vannak jelenleg telepítve:

   ```
   cimprovider -l -s
   ```

   - Keressük ki, hogy milyen IP-vel kapcsolatos providerek vannak (használjuk a grep parancsot).

5. OpenPegasus által használt port kiderítése

   Nézzük meg, hogy milyen portokon figyel jelenleg a virtuális gép (a `-t` a TCP kapcsolatokat jeleníti meg, a `-l` a listening állapotban lévőket, a `-p` a hozzá tartozó folyamatot keresi ki):

   ```
   sudo netstat -t -l -p
   ```

- Keressük ki a cimserver folyamathoz tartozó port számát (a --numeric hatására numerikus formában jeleníti meg az ismert portokat is, különben a `/etc/services` fájlban találjuk meg a megfeleltetést).

6. Alap osztály lekérdezése helyben

   Kérdezzünk le egy olyan osztályt, ami biztos létezik:

   ```
   wbemcli gc 'http://localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   A `gc` a `GetClass` művelet rövidítése.

   Az előbb a *netstat* kimenetéből kiderült, hogy például az 5989-es porton (ez a wbem-https port) és az 5988-as porton is figyelünk (ez a wbem-http port). Amíg tesztelünk és hibát keresünk, érdemes a http portot használni, hogy például Wiresharkban meg tudjuk nézni a forgalmat, azonban ha összeállt már a rendszer, érdemes kipróbálni https használatával is, éles rendszerben annak a használata a javasolt.

   Az objectPath-ban meg kell adni a névteret (`root/cimv2`), majd egy kettőspont után az osztály nevét (`CIM_OperatingSystem`).

   A válasz erre az, hogy hitelesíteni kell magunkat, így adjunk meg felhasználónevet és jelszót is:

   ```
   wbemcli gc 'http://pegasus:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   A pegasus felhasználó az OpenPegasus telepítésével létrejövő helyi felhasználó.

   Hogy a kimenet olvashatóbb legyen, adjuk meg az `-nl` paraméter (new line, új sorokat szúr be az egyes tulajdonságok után).

   ```
   wbemcli -nl gc 'http://pegasus:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   Még annyit nézzünk meg, hogy a háttérben milyen üzeneteket küld és kap a CIM-XML kliensünk. Erre a `-dx` kapcsoló való.

   - Keressük ki a kapott üzenetekből, hogy a TotalSwapSpaceSize tulajdonság milyen típusú!

   Kérdezzük le az osztály példányait is. Erre az `ei`, `EnumerateInstances` művelet használható.

   ```
   wbemcli -nl ei 'http://pegasus:LaborImage@localhost:5988/root/cimv2:CIM_OperatingSystem'
   ```

   - A kimenetben keressük meg, hogy mikor bootolt fel az operációs rendszer (LastBootUpTime tulajdonság)!

7. Lekérdezhető példányok listája

   Nézzük meg, hogy a rendszer providerei milyen CIM osztályokat nyújtanak:

   ```
   wbemcli -nl ei
   'http://pegasus:LaborImage@localhost:5988/root/PG_InterOp:PG_ProviderCapabilities
   ' | grep ClassName
   ```