Konfigurációkezelési technológiák

Gyakorlati útmutató
Készítette: Micskei Zoltán
Utolsó módosítás: v1.3.1, 2013.04.04.

A segédlet célja, hogy bemutassa a konfigurációkezelési technológiákhoz tartozó alap eszközöket, és hogy megismerkedjünk a *Common Information Model*lel (CIM).

Figyelem:

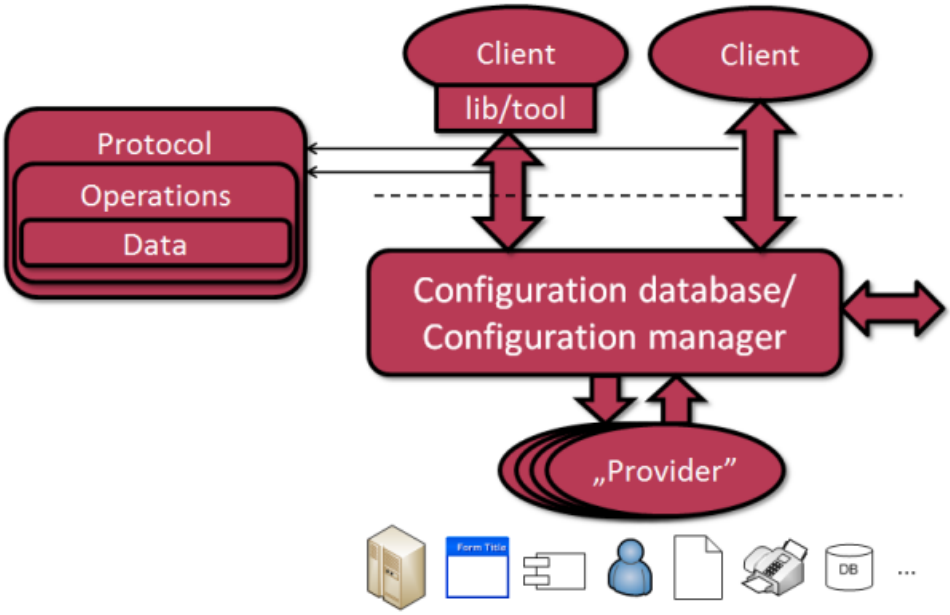
- Az utasításokat ne másoljuk, hanem tényleg gépeljük is be. Különben nem rögzül a szintaktika.
- A gyakorlat elvégzése előtt nézzük át a kapcsolódó két előadást!

Tartalom

1 Konfigurációkezelési szabványok és technológiák

A konfigurációkezelés terület célja többek között az, hogy legyen egy központi nyilvántartásunk az informatikai rendszerünk elemeiről és az azok közötti kapcsolatokról. Cél továbbá, hogy ez lehetőleg egy közös, könnyen hozzáférhető és akár beavatkozásra is képes felületet nyújtson a teljes rendszer minden szintjéről (a hardvererőforrások állapotától kezdve az operációs rendszer beállításán át a futtatott szolgáltatások komponenseivel bezárólag).

Egy általános konfigurációkezelési architektúrát ábrázol az alábbi ábra (1. ábra).

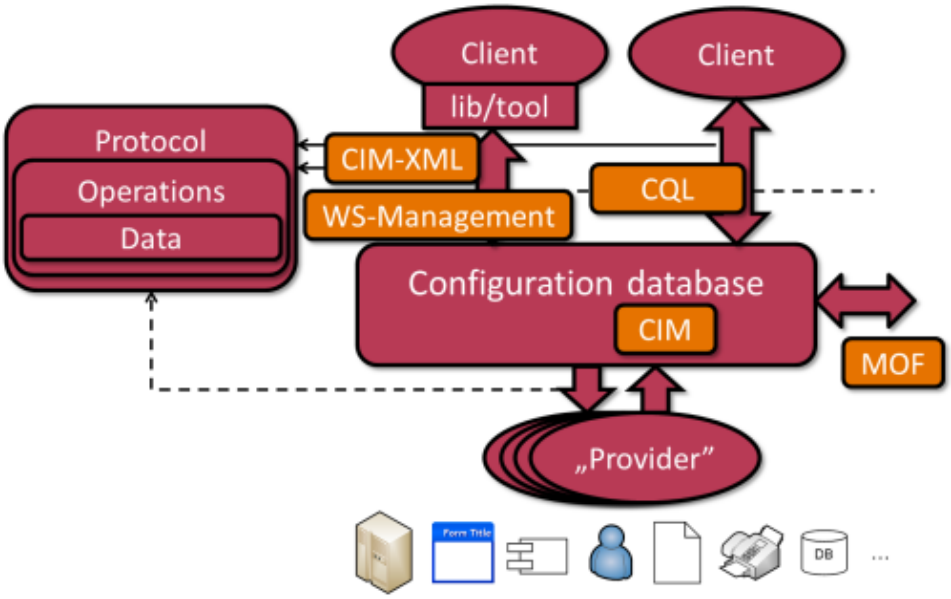


1. ábra: Konfigurációkezelés általános architektúrája

Az architektúrában szereplő főbb elemek a következőek.

- *Menedzselt elemek*: az ábrán alul szerepelnek felügyelt informatikai rendszert alkotó elemek. Ezek lehetnek hardverelemek (szenzor, memóriamodul, CPU…), az operációs rendszer és fogalmai (eszközmeghajtók, futó folyamatok, telepített csomagok…), kiszolgálóprogramok (webkiszolgáló és moduljai, adatbázis-kiszolgáló) vagy akár saját szolgáltatások és alkalmazások.
- *Szolgáltató (provider)*: az a szoftver komponens, ami az információt begyűjti és továbbítja, valamint a beavatkozást elvégzi a menedzselt elemen.
- *Konfigurációs adatbázis (configuration database)*: a konfigurációs adatokat tároló és azok lekérdezését lehetővé tévő adatbázis. Ehhez biztosítani kell valami import/export funkciókat megvalósító felületet is.
- *Távoli elérési felület*: a szabványos és platform független eléréshez definiálni kell, hogy milyen hordozóprotokollt kell használni, milyen műveleteket lehet elvégezni az adatbázison és hogyan kell az átvitt adatokat becsomagolni.

Ezekhez a komponensekhez a DMTF nevű szervezet által kidolgozott ajánlásokkal fogunk foglalkozni a tantárgy keretein belül. A számunkra fontos ajánlásokat és technológiákat az alábbi ábra szemlélteti (2. ábra).



2. ábra: Konfigurációkezelési ajánlások és technológiák

A fontosabb ajánlások és technológiák:

- *Common Information Model (CIM)* A CIM ajánlás a következő részeket foglalja magába:
  - *CIM Metamodel*: definiál egy modellezési nyelvet, amivel konfigurációs modelleket lehet megadni.
  - *CIM Schema*: létrehoztak egy rendkívül részletes, hierarchikus, bővíthető modellt, amivel informatikai rendszereket lehet leírni.
  - *Managed Object Format (MOF)*: CIM modellekhez kidolgozott szöveges import/export formátum, amit a CIM-kompatibilis eszközöknek támogatni kell.
  - *CIM Object Manager (CIMOM):* a CIM-alapú konfigurációs adatbázisokat CIMOM-nak nevezik. A szabvány ezek belső megvalósításával kapcsolatban nem köt ki semmit.
- *Web Based Enterprise Management* (WBEM) Ajánlásgyűjtemény, ami a CIMOM-ok távoli eléréséhez fogalmaz meg protokollokat:
  - *CIM-XML*: HTTP-re épülő protokoll, mely a CIM adatokat XML-formátumba csomagolja be.
  - *WS-Management*: SOAP-alapú webszolgáltatásokra épülő általános távoli menedzsmentet megvalósító nyelv, amivel CIMOM-okat is lehet kezelni, a CIM-alapú modelleket le lehet erre a formátumra is képezni.
  - *CIM Quary Language (CQL)*: az SQL-hez hasonlító lekérdezőnyelv CIMOM-okhoz.

(Ezek most csak a legfontosabb DMTF ajánlások voltak, van még több is.)

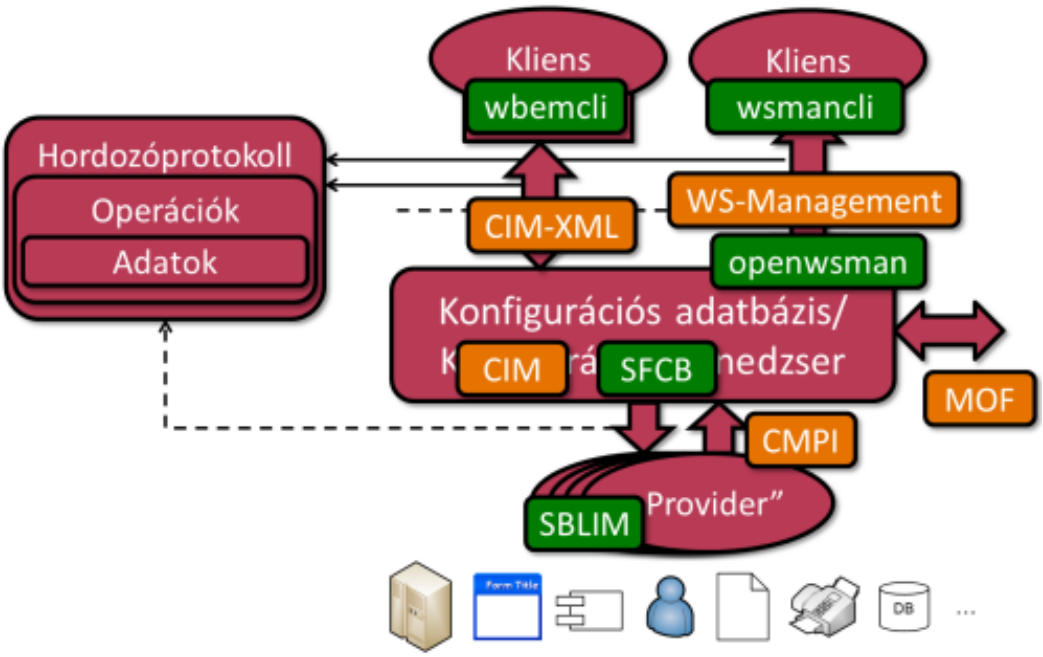
A gyakorlat során ezeknek a technológiáknak a megvalósításait nézzük meg Linux és Windows platformokon.

2 Linux: sblim-sfcb, wbemcli és openwsman

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

- CIM Object Manager: sblim-sfcb[1] 1.4.3-4.1
- CIM-XML kliens: wbemcli[2] 1.6.2
- Providerek: sblim-cmpi-*
- WS-Management szerver: openwsman[3] 2.3.7-83.2
- WS-Management kliens: wsmancli 2.3.0-39.2

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



3. 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/

2.1 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 belépési információ a gép könyvtárában lévő README fájlban található.

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, továbbá a beállítások leírását megtaláljuk az `sfcbd` manual oldalán is.

3. sfcb elindítása

   Kérdezzük le, hogy fut-e a CIM kiszolgáló:

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

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

   ```
   sudo /etc/init.d/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 -k -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_Network.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 dokumentációjában[4].

2.2 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 (mostantól a példákban ezt a felhasználónevet és jelszót használjuk, ha az adott rendszeren mások az adatok, akkor értelemszerűen módosítsunk majd minden későbbi parancsot):

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

   Lehet, hogy nem fogadja el, és ilyen hibaüzenetet dob:

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

   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 -A sfcb meres
   ```

   Ha most megismételjük, akkor most már sikeres a lekérdezés.

[4] SBLIM wiki. SfcbTheBook. URL: http://sourceforge.net/apps/mediawiki/sblim/index.php?title=SfcbTheBook

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 (az előbbi feladatban ezt már megtaláltuk). Adjunk ki egy, a következőhöz hasonló parancsot[5] (csak írjuk át benne az attribútumok értékét az aktuális gép adatainak megfelelően):

```
wbemcli -nl gi
'http://meres:LaborImage@localhost:5988/root/cimv2:Linux_OperatingSystem.Name="linux.site",Cr
eationClassName="Linux_OperatingSystem",CSName="linux.site",CSCreationClassName="Linux_Comput
erSystem"'
```

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, például:

```
*
* wbemcli: Cim: (1) CIM_ERR_FAILED: Could not get CS/OS CreationClassName of instance.
*
```

Ilyenkor érdemes ellenőrizni, hogy mik az adott osztály kulcsai.

3. Kapcsolódó példányok lekérdezése

[5] Figyelem: a segédletben a legtöbbször egy-egy parancsot fogunk kiadni, de ezek általában olyan hosszúak, hogy nem férnek ki egy sorba. Ettől függetlenül még egy sorba kell beírni őket.

Ha lekérdeztünk egy adott példányt, akkor megkereshetjük, hogy különböző asszociációkon keresztül milyen másik példányok kapcsolódnak ehhez. Erre az `ain` (association instance names) művelet használható.

```
wbemcli ain
'http://localhost/root/cimv2:Linux_ComputerSystem.CreationClassName="Linux_ComputerSystem",Na
me="linux.site"'
```

Itt most egy másik osztálynak, a Linux_ComputerSystem egy példányának a kapcsolatait kértük le. Ha elírnánk a példány kulcsainak értékét, akkor például a következő hibaüzenetet kapjuk:

```
*
* wbemcli: Cim: (6) CIM_ERR_NOT_FOUND: Source object not found.
*
```

Ha pedig nem adjuk meg az összes kulcsának értékét, akkor ez lehet a hibaüzenet:

```
*
* wbemcli: Cim: (1) CIM_ERR_FAILED: GetInstance of source object failed.
*
```

4. 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'
   ```

5. 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'
   ```

   Erre a wbemcli verziójától függően többféle hibaüzenet is jöhet. Az egyik:

   ```
   *
   * wbemcli: Http Exception: Problem with the SSL CA cert (path? access rights?)
   *
   ```

   A másik hibaüzenet:

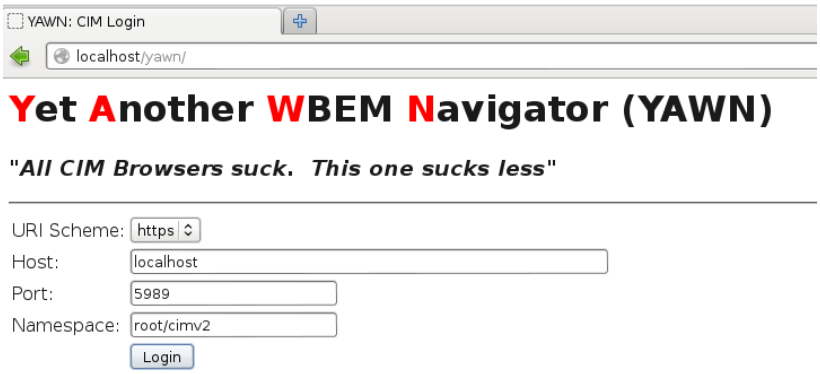
   ```
   *
   * wbemcli: Http Exception: Peer certificate cannot be authenticated with known CA
   certificates
   *
   ```

   Az sfcb kiszolgáló által felkínált tanúsítványt a helyi gépünk nem fogadta el hitelesnek, mert nem egy megbízható hitelesítés-szolgáltató (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 (elvileg, ha nem adtuk meg a `-noverify` kapcsolót, akkor ezt kötelező lenne megadni, de az eszköz nem ellenőrzi).

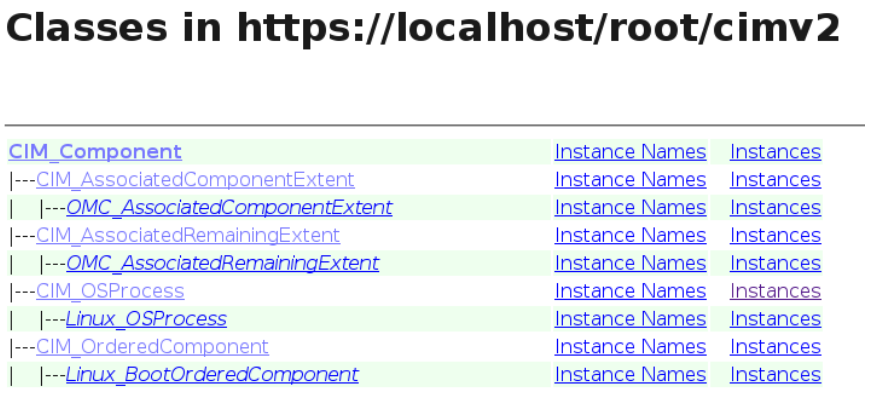
2.3 YAWN

A Yet Another WBEM Navigator (YAWN) egy egyszerű webes felület, aminek a segítségével egy CIMOM tartalmát meg tudjuk nézni.



4. ábra: YAWN belépőképernyő

Bejelentkezés után kilistázza az összes, a CIMOM-ban definiált osztályt, feltüntetve az öröklődési viszonyokat is (5. ábra).



5. ábra: CIM-osztályok listája a YAWN felületén

Ha megnézünk egy adott osztályt, akkor kilistázza az eszköz a tulajdonságait és metódusait azok leírásával együtt. Lekérdezhetjük egy osztály példányait is.

Próbáljuk ki a YAWN eszközt, és ismerkedjünk meg a CIMOM tartalmával:

- Nézzük meg a definiált CIM osztályokat, nézzük meg néhánynak a tulajdonságait.
- Kérdezzük le néhány osztály példányait. Figyelem, sok osztályhoz nincs megfelelő provider, ilyenkor „Provider not found or not loadable" választ fogunk visszakapni. (Tipikusan a Linux kezdetű osztályokhoz van provider).
- Próbáljuk lekérdezni egy kapcsolóosztály példányait (pl. Linux_CSProcessor)! Hogyan hivatkoznak a kapcsolódó példányokra?
- (Sajnos az adott verzióban a felületen lévő *Associated Classes* linkek tipikusan nem működnek.)

2.4 Távoli lekérdezés wbemcli segítségével

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

1. Távoli gép létrehozása

   Készítsünk egy klónt vagy másoljuk le a 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 openSUSE esetén:

   ```
   /etc/HOSTNAME
   ```

   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'
   ```

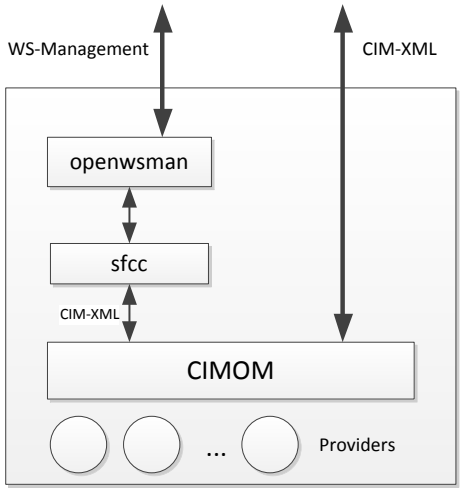
   Nyilván ehhez az is kell, hogy a megfelelő port ki legyen engedve a tűzfalon. A kiadott virtuális gépen ki van kapcsolva a tűzfal, ez éles rendszerben nem engedhető meg.

2.5 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.)

Az alábbi ábra szemlélteti az openwsman és a szükséges komponensek viszonyát (6. ábra). Az openwsman az SFCC[6] (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).

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



6. ábra: openwsman architektúrája

1. openwsman kiszolgáló beállítása

   Az openwsman a beállításait a következő konfigurációs fájlban tárolja:

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

   (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[7] találhatóak információk a konfigurációs beállításokról.

   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.

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

```
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
```

(Ez a kiadott virtuális gépen most nem működik.)

2.6 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
```

Válaszként egy XML részletet, egy WS-Management üzenetet kapunk, amiből kiderül a kiszolgáló néhány adata. Így most már végre tudunk hajtani egy egyszerű kérést. Ha a `hostname` paramétert lecseréljük, akkor akár távolról is meg tudjuk szólítani az openwsman kiszolgálónkat.

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'
```

Figyelem: ehhez futnia kell az sfcb-nek is a helyi gépen, hisz az openwsman attól kéri le az adatokat!

(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. A parancsban szereplő \ csak jelzi a Bash-nak, hogy folytatódik majd a sor.)

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/InsufficientSelecto
   rs</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 kulcs tulajdonságát 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",DeviceID="0",Sy
   stemCreationClassName="Linux_ComputerSystem",SystemName="irfserver.irf.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=L
   inux_ComputerSystem,SystemName=irfserver.irf.local'
   ```

   Az eredményként vissza is kapjuk a keresett példányt.

   Ha elírjuk az objektum kulcsainak 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_ComputerSystem,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 97 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"
```

(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 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,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.

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 meres -p LaborImage \
 enumerate 'http://schemas.dmtf.org/wbem/wscim/1/*' \
 --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, akkor ResourceURI-nak a DMTF „All Classes" URI-ját adjuk meg, ahogy az a fenti példában is szerepel[8].

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 itt is 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:

```
<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_ComputerSyst
em'
```

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

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_ComputerSyst
em,AssociationClassName=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

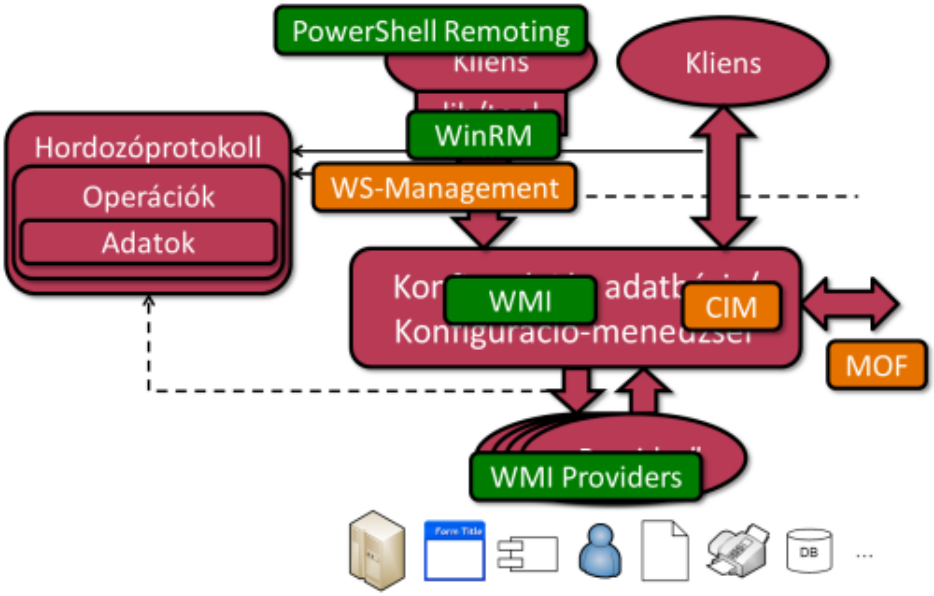
```
Provider not found or not loadable
```

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.

3 Windows: WMI, WinRM

A feladatokat egy Windows 8 virtuális gépen fogjuk végrehajtani. A Windows 8 már alapból tartalmazza a *Windows Management Framework* (WMF) 3.0-s verzióját. Ez többek között a CIMOM-ot (WMI Object Manager), a WS-Management klienst és kiszolgálót (WinRM) és az ezek kezeléséhez szükséges PowerShell 3.0-s cmdleteket foglalja magába. Így telepíteni további komponenseket nem kell, a feladatunk az lesz, hogy ezeket megfelelően beállítsuk.

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



7. ábra: Konfigurációkezelési technológiák Windowsra

3.1 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*[9] kis segédprogramot letölteni.

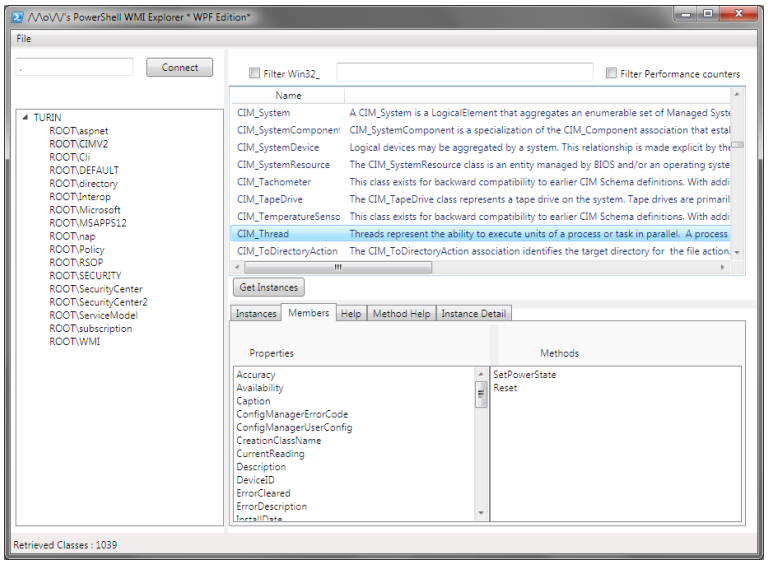
   Figyelem: A letöltés után a fájl tulajdonságlapján az **`Unblock`** megnyomásával oldjuk fel a távoli letöltés miatti zárolást.

   Indítsuk el a letöltött szkriptet:

   ```
   .\WpfWmiExplorer.ps1
   ```

   A következőhöz hasonló kép fogad majd minket:

[9] http://thepowershellguy.com/blogs/posh/pages/powershell-wmi-explorer.aspx (ez most átmenetileg itt érhető el: http://mit.bme.hu/~micskeiz/files/WpfWmiExplorer.zip)



8. 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. CIM/WMI kezelésére szolgáló cmdletek

Keressük meg a CIM/WMI kezelésére szolgáló cmdleteket:

```
Get-Command -Module CimCmdlets
```

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

Nézzük néhány példát a `Get-CimInstance` használatára:

```
Get-Help Get-CimInstance -examples
```

Ezzel nagyjából képet lehet kapni, hogy mit tud a `Get-CimInstance`. 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-CimInstance -full | more
```

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

```
Get-CimInstance CIM_Memory
```

Válaszként `Microsoft.Management.Infrastructure.CimInstance` típusú objektumokat kapunk vissza, amiknek a CIM osztálynak megfelelő tulajdonságai vannak. Ez alapján már könnyen tudunk szűrni vagy rendezni PowerShellben a korábban tanult módon.

```
Get-CimInstance 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.

A példányokon kívül maguk a CIM osztályok adatait is lekérdezhetjük:

```
Get-CimClass CIM_Memory
```

Nézzük meg, hogy a visszakapott osztálynak milyen tulajdonságai vannak:

```
(Get-CimClass CIM_Memory).CimClassProperties
```

Lehet az osztály nevében wildcard karaktert is használni:

```
Get-CimClass -ClassName *disk*
```

- A visszaadott osztályok közül melyikből tudhatjuk meg, hogy mekkorák a lemezeken lévő partíciók?

Sőt, lehet minősítőre is keresni (vagy akár másik paraméterrel tulajdonságra vagy metódusra is):

```
Get-CimClass -QualifierName "Abstract" -Namespace root\standardcimv2
```

A fenti parancs a megadott névtérben keresi meg az absztrakt minősítővel ellátott osztályokat.

- Keressük meg, hogy a root\cimv2 osztályban milyen CIM kezdetű asszociációs osztályok vannak jelenleg!

3. Szűrés a lekérdezésben

Szűrésre több lehetőségünk is van. Alapvetően CIM Query Language (CQL)[10] és WQL (WMI Query Language)[11] lekérdezéseket fogalmazhatunk meg. A `-Filter` paraméter használatával csak a lekérdezés WHERE részét kell megadni:

```
Get-CimInstance -ClassName CIM_Process -Filter "WorkingSetSize > 100000000"
```

Arra figyeljünk, hogy ha CIM-osztályt kérünk le, akkor a szűrőben csak olyan attribútum szerepelhet, ami abban is definiálva van, és például a Win32 megfelelőjében definiált extrák nem.

```
# produces error, as HandleCount is defined only in Win32_Process
Get-CimInstance -ClassName CIM_Process -Filter "HandleCount > 1000"
```

A másik lehetőség, hogy a `-Query` paraméterben a teljes lekérdezést megadjuk. Ilyenkor azonban nem adhatjuk már meg a `-ClassName` paramétert:

```
Get-CimInstance -Query "Select name, state FROM Win32_Service WHERE StartMode = 'Auto'"
```

[10] A WMI jelenlegi verziója nem támogatja a CQL értékét a QueryDialect paraméternek, csak a WQL-t.
[11] A WQL zintaxisát lásd az `about_WQL` súgótémában.

A WQL segítségével tehát bonyolultabb lekérdezéseket is meg tudunk fogalmazni.

4. 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 a `Get-CimAssociatedInstance` cmdletet használni erre. Alapesetben ez egy CIM példányt vár paraméterként, amihez megkeresi az összes kapcsolódó példányt:

```
# get an instance
$c = Get-CimInstance -ClassName CIM_Processor -Filter 'DeviceID = "CPU0"'
# get the instances associated to this one
Get-CimAssociatedInstance -InputObject $c
```

   Lehetőségünk van arra is, hogy csak bizonyos típusú példányokat kapjunk vissza:

```
Get-CimAssociatedInstance -InputObject $c -ResultClassName CIM_ComputerSystem
```

   A másik lehetőség a kapcsolódó példányok lekérdezésére a WQL nyelv `ASSOCIATORS OF` kulcsszavának használata [7]. Például így kérdezhetjük le egy adott *Win32_LogicalDisk* példányhoz tartozó összes kapcsolódó példányt.

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

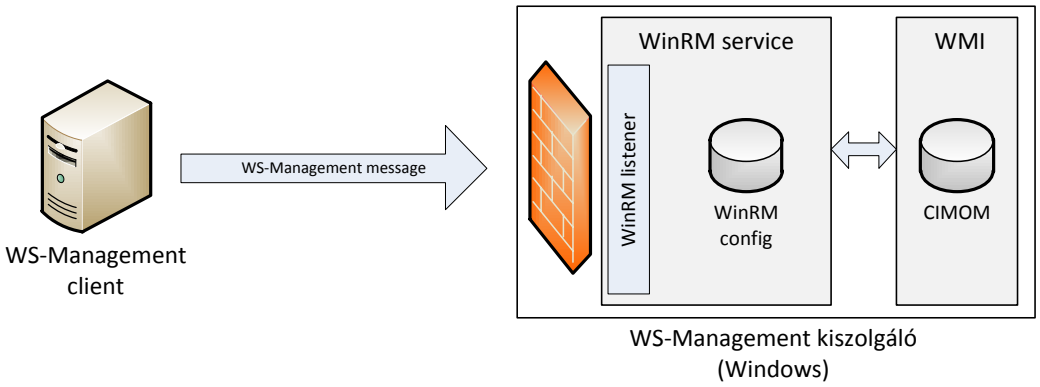
   Ez a parancs is megvizsgálja az összes kapcsolóosztályt, így kapunk például *Win32_ComputerSystem*, *Win32_DiskPartition* és *Win32_Directory* példányt is. Ha csak egy konkrét kapcsolatra vagyunk kíváncsiak, akkor az `AssocClass` szűrőfeltételt lehet használni.

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

3.2 WinRM használata

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

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.



9. ábra: A WinRM architektúrája

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 aktuálisan telepített PowerShell és WinRM verziót így tudjuk ellenőrizni:

   ```
   $PSversiontable
   ```

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

   ```
   WinRM Quick Configuration
   Running the Set-WSManQuickConfig command has significant security implications, as it enables
   remote management through the WinRM service on this computer.
   This command:
    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 WS-Management traffic.
    5. Enables Kerberos and Negotiate service authentication.
   ```

   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 használjuk a `Set-WSManQuickConfig` cmdlet `-SkipNetworkProfileCheck` paraméterét. Ebben az esetben a publikus hálózatokra olyan tűzfalszabályt állít be, amivel csak az adott lokális hálózatból lehet elérni a WinRM szolgáltatást.

Ha ellenőrizni akarjuk, hogy sikerül-e távolról csatlakozni, akkor használjuk a *Test-WSMan* cmdletet. Ez a WS-Management IDENTIFY műveletét használja (a példákba innentől kezdve az általunk használt távoli gép IP-címét és bejelentkezési adatait helyettesítsük be).

```
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 segítéségével

Vegyük elő valamelyik szokásos CIM osztályunkat, és kérdezzük le a példányait WS-Management segítéségével. Az alacsonyszintű lekérdezésekhez 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. (Figyelem, itt még Windows-specifikus URI-t használtunk a lekérdezésben, ez Linuxot futtató távoli gép esetén nem működne!)

```
$ip = "192.168.21.151"
Get-WSManInstance wmicimv2/Win32_Processor -ComputerName $ip -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 WSMAN: PowerShell provider segítségével kezelhetjük[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 8 gépen a következőt kapjuk:

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

Name                    Value
----                    -----
MaxEnvelopeSizekb       500
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"
```

[12] További 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.

(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:

```
Get-WSManInstance wmicimv2/Win32_Processor -ComputerName $ip -Credential meres -Enumerate
```

Itt most megadtuk a `-Credential` paraméterben a távoli gépen használandó felhasználónevet. Hogy ne kelljen minden egyes későbbi parancshoz begépelni külön a jelszót, mentsük el egy változóba:

```
$c = Get-Credential
```

3. Szabványos URI használata

   Az előző lekérdezés csak Windows rendszert futtató távoli géppel működhet, hisz az erőforrás URI-jánál a Microsoft-specifikus változatot használtuk. Írjuk át ezt a DMTF által szabványosítottra:

   ```
   Get-WSManInstance cimv2/CIM_Processor -ComputerName $ip -Credential $c -Enumerate
   ```

   Erre a következő választ kapjuk:

   ```
   The WS-Management service cannot process the requst. The class CIM_Processor does not exist
   in the root/hardware namespace.
   ```

   A probléma az, hogy a WS-Management szabvány nem specifikálja, hogy melyik legyen az alapértelmezett névtér, és a WinRM cmdletek történelmi okok miatt a `root/hardware` névteret használják. Jelezhetjük, hogy más névteret szeretnénk használni:

   ```
   Get-WSManInstance cimv2/CIM_Processor?__cimnamespace=root/cimv2 `
    -ComputerName $ip -Credential $c -Enumerate
   ```

   Így már működik a szabványos lekérdezés is (ez viszont alapvetően csak Windows 8-as távoli gépen működik, Windows 7 esetén további gondok lennének még [6]).

4. 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 segíthet például a következő lekérdezés:

   ```
   (Get-CimClass CIM_NetworkAdapter).CimClassProperties | select Name, Qualifiers
   ```

   Itt azokat a tulajdonságokat kell keresni, amik `key` minősítővel rendelkeznek. A kulcsok neveit és értékeit egy hash-táblába kell összegyűjteni és átadni a `SelectorSet` paraméternek:

   ```
   Get-WSManInstance -ComputerName $ip -Credential $c `
    -ResourceURI wmicimv2/Win32_NetworkAdapter -SelectorSet @{DeviceID=1}
   ```

5. 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 $ip -Credential $c -Enumerate -Dialect Selector `
 -ResourceURI wmicimv2/Win32_NetworkAdapter `
 -Filter 'AdapterType="Ethernet 802.3";Speed="1000000000"'
```

(Arra figyeljünk, hogy itt az egyes tulajdonságokat 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 a windowsos „All Classes" URI-t kell használni:

```
Get-WSManInstance -ComputerName $ip -Credential $c -ResourceURI wmicimv2/* -Filter `
 "SELECT name, index FROM Win32_NetworkAdapter WHERE ServiceName = 'tunnel' AND Index > 5" `
 -Dialect WQL -Enumerate
```

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 [8] 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 [9] 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 $ip -Credential $c -Enumerate -ResourceURI wmicimv2/* `
 -Dialect Association -Filter "{object=Win32_LogicalDisk?deviceid=C:}" -Associations
```

Figyelem: kapcsolatok lekérdezésekor a kliensen Windows 8 előtt 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 objektumot a következő tulajdonságok mentén elérjük:

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

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

A másik lehetőség, hogy a konkrét példányokat kérjük vissza:

```
Get-WSManInstance -ComputerName $ip -Credential $c -ResourceURI wmicimv2/* -Filter `
 "{object=Win32_LogicalDisk?deviceid=C:;AssociationClassname=Win32_LogicalDiskToPartition}" `
 -Dialect Association -Enumerate
```

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 [9], hogy különböző elemeket használhatunk a szűrőben. Másik tipikus probléma, ha nem jól adtuk meg az object részben a kért osztály azonosítóját, és nem találja a távoli fél az adott példányt.

3.3 Távoli CIM osztályok lekérdezése WinRM segítségével

Az előző részben a WinRM alacsony szintű cmdleteinek a használatát néztük meg. Ezzel elvileg bármilyen WS-Management erőforrást el tudunk érni, és minden protokoll szintű részletet tudunk szabályozni. Szerencsére a WMF 3.0-s verziójában már az új CIM cmdletek alapértelmezés szerint WS-Managementet használnak távoli lekérdezésre, és sok alsóbb szintű részletet elfednek.

1. Egyszerű távoli lekérdezés

   A Get-CimInstance cmdletnek van egy -ComputerName paramétere, amivel egyszerűen lehet távoli lekérdezést végrehajtani.

   ```
   $ip = "192.168.21.151"
   Get-CimInstance -ClassName Win32_DiskDrive -ComputerName $ip
   ```

   Ez azonban sok esetben még így nem elég, mert például ez a helyi felhasználónevet és jelszót próbálja meg felhasználni a távoli gépen.

2. Munkamenetek (session) használata

   A távoli géphez való kapcsolódás összes beállítását egy úgynevezett munkamenetbe lehet összefoglalni. Hozzunk egy ilyet létre (feltételezve, hogy a távoli gép Windows 8-at futtat):

   ```
   $s = New-CimSession -Authentication Default -Credential meres -ComputerName $ip
   ```

   Ezt mostantól kezdve az összes CIM cmdletben felhasználhatjuk:

   ```
   Get-CimInstance -ClassName Win32_DiskDrive -CimSession $s
   ```

   Működik így például a kapcsolódó példányok egyszerű lekérdezése:

   ```
   $d = Get-CimInstance -ClassName Win32_DiskDrive -CimSession $s
   Get-CimAssociatedInstance -InputObject $d -CimSession $s
   ```

(Így azért nagyságrenddel egyszerűbb, mintha a `Get-WsManInstance` cmdletet használtuk volna.)

TIPP: lehetőség van a munkamenet létrehozásakor több távoli számítógépet is megadni.

3. Szűrés a távoli lekérdezésben

   Arra figyeljünk oda, hogy távoli lekérdezés esetén csak a szükséges adatokat kérdezzük le, és inkább a távoli gépen szűrjünk. Erre való a korábban megismert `-Filter` vagy `-Query` paraméter. Például a korábban kipróbált lekérdezés így néz ki, ha távoli gépen hajtjuk végre:

   ```
   Get-CimInstance -Query "Select name, state FROM Win32_Service WHERE StartMode = 'Auto'" `
    -CimSession $s
   ```

   Azt érdemes még kipróbálni, hogy az új cmdletek implementálják a PowerShell cmdletek általános paramétereit, így például használhatjuk a `-Verbose` kapcsolót a kérés részleteinek megjelenítésére.

4 Platformok közötti lekérdezések

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

4.1 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

```
IP=192.168.21.151
wsman identify -h $IP
```

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)[14].

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 most be kell kapcsolni[15]:

```
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:

[14] Az újabb openwsman verziók tudják már a Negotiate protokollt is kezelni, az elvileg használható lenne.
[15] Ha van olyan hálózati kapcsolata a gépnek, ami Public típusú, akkor ez nem fog menni. Ilyenkor ezt a lokális Group Policy-ből lehet beállítani az ellenőrzés megkerülésével.

```
wsman identify -h $IP --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[16]. Sokat segíthet az is, ha `-d 6` kapcsolóval indítjuk a *wsman* klienst.

2. Adatok lekérdezése

Nézzünk akkor kezdésnek egy ENUMERATE műveletet:

```
wsman enumerate 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor' \
 -h 192.168.21.151 --auth basic -u meres -p LaborImage
```

Egy dologra kell figyelni. Az erőforrás URI-ban itt a DMTF-specifikus prefixet használtuk. A DMTF által szabványosított prefix használata nem megy tökéletesen Windows 8 előtt [6]. 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 $IP --auth basic -u meres -p LaborImage -N root/cimv2 enumerate
http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/CIM_Processor
```

Nézzünk meg egy GET műveletet is:

```
wsman get 'http://schemas.dmtf.org/wbem/wscim/1/cim-schema/2/Win32_ComputerSystem?Name=windows8-
base' -h $IP --auth basic -u meres -p LaborImage -N root/cimv2
```

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.

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.

[16] 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

4.2 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 és titkosítatlan kapcsolatot a WinRM kliens beállításaiban.

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

   Ha csatlakozni próbálnánk, akkor a TrustedHosts beállításra panaszkodik, így állítsuk be azt a 3.2 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-kiszolgáló 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) URI prefixeket és osztályneveket kell használni.

4. Új CIM cmdletek használata

   A 3.1 szakaszban megismert CimCmdlet modul parancsai használhatóak nem Windowst futtató gépek lekérdezésére is. Ehhez csak egy megfelelően felparaméterezett CIM munkamenetet kell használni:

   ```
   $s = New-CimSession -Authentication Basic -Credential meres -ComputerName 192.168.21.150
   Get-CimInstance -CimSession $s -ClassName CIM_Fan
   ```

5. Szűrés a távoli gépen

A távoli gépen való szűréshez használhatunk például egy CQL lekérdezést. Legnagyobb meglepetésünkre mindenféle mágia használata nélkül működik, a PowerShell implementáció szabványos URI-kat generál a háttérben, jól állítja be az alapértelmezett névteret, és a lekérdezés pont a keresett példányt adja csak vissza:

```
Get-CimInstance -CimSession $s -QueryDialect CQL `
 -Query "SELECT * FROM CIM_Fan WHERE Name = 'FAN0'"
```

6. Kapcsolódó példányok lekérdezése

A kapcsolódó példányokat hasonlóan kaphatjuk meg, mint a korábbi feladatokban:

```
$i = Get-CimInstance -CimSession $s -QueryDialect CQL `
 -Query "Select * FROM CIM_Fan WHERE Name = 'FAN0'"
# query associated instances
Get-CimAssociatedInstance -InputObject $i -CimSession $s
```

A visszakapott objektumok `CimClass` tulajdonságából derül ki, hogy ténylegesen milyen példányok.

5 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 távolról az újabb verziókban már a WinRM által biztosított WS-Management felületen lehet elérni. A lekérdezésekhez PowerShell cmdleteket használtunk.

6 További információ

Általános szabványok

[1] DMTF. „Web Services for Management (WS Management)”, 1.1.1, DSP0226, 2012. URL: http://www.dmtf.org/sites/default/files/standards/documents/DSP0226_1.1.1.pdf

[2] DMTF. „WS-Management CIM Binding Specification”, 1.2.0, DSP0227, 2011. URL: http://www.dmtf.org/sites/default/files/standards/documents/DSP0227_1.2.0.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, 2011. URL: http://linux.dell.com/files/whitepapers/WBEM_based_management_in_Linux.pdf

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

Windows

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

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

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

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

7 Függelék

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

7.1 ECUTE

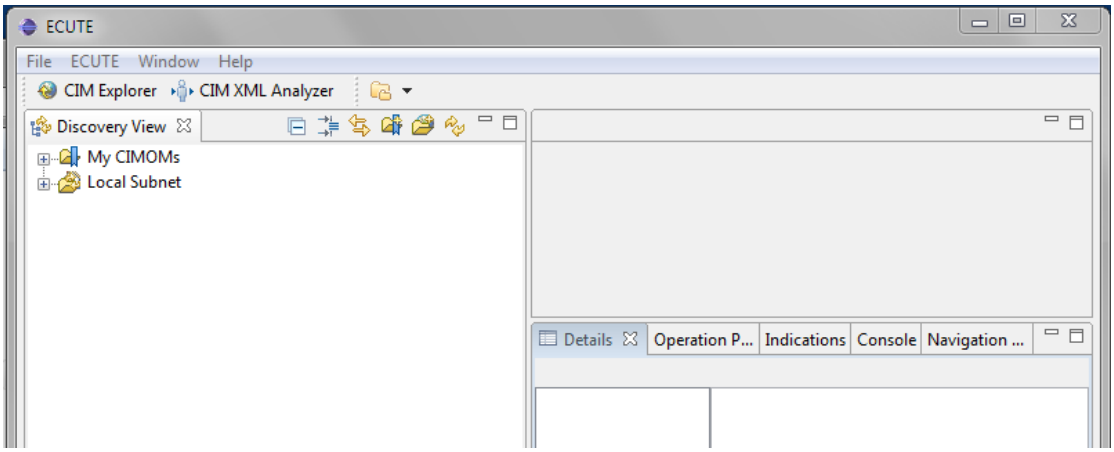
Az ECUTE (Extensible CIM UML Tooling Environment)[17] az SBLIM projekt része. Egy Eclipse alapú GUI-t biztosít CIMOM-ok megnézésére. Az adatokat CIM-XML segítségével kérdezi le. SLP (Service Location Protocol) segítségével fel is tudja deríteni, hogy milyen CIMOM-ok vannak a helyi hálózaton.

A következő részekből áll:

- *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[18].

Elindítás után a következő kép fogad.



10. ábra: ECUTE képernyő

Az Explorer használatához először hozzá kell adni a *My CIMOMs* részhez egy új elemet (11. ábra).

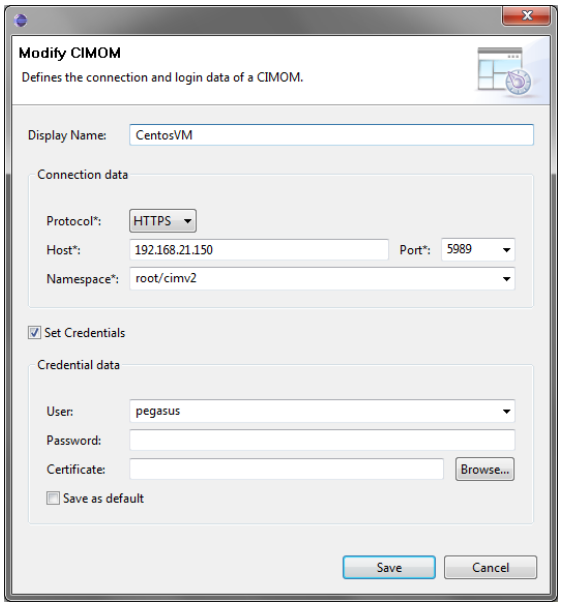
Ezután már tudunk osztályneveket lekérdezni, az osztályok tulajdonságait megjeleníteni, valamint példányokat felsorolni. Az ECUTE súgója elég használható, ha elakadunk, azt érdemes megnézni.

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

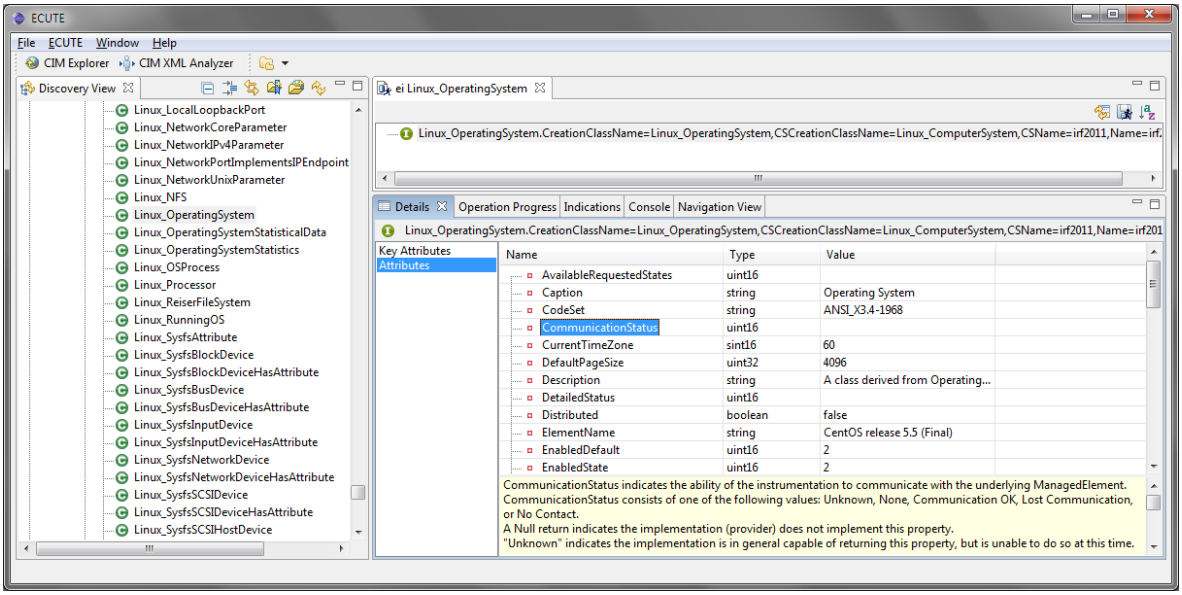
[18] 1.7-es Java használata esetén már nem biztos, hogy elindul az ECUTE RCP, mert annyira régi Eclipse verziót használ. Ilyen esetben egy újabb Eclipse-be kell telepíteni a letöltött Analyzer és Explorer feature-t (*Install New Software...* menüpont, majd *Local* kiválasztása az Update Site résznél), majd az *Open Perspective* menüben megtalálhatóak az ECUTE Explorer és Analyzer perspektívák.

A következő képen a `Linux_OperatingSystem` példányait kérdeztük le, majd annak egy tulajdonságát nézzük meg (12. ábra).

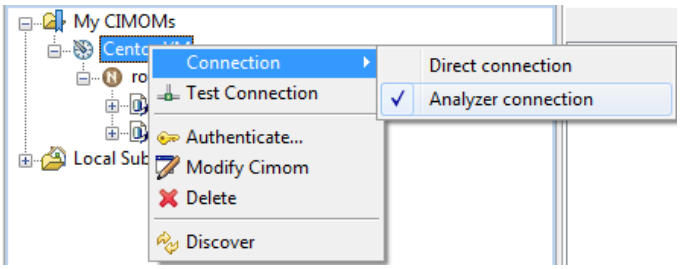
Próbáljuk ki most az *Analyzer* komponenst is. Ehhez a CIMOM tulajdonságainál a kapcsolat típusát állítsuk át *Analyzer Connection* értékre (13. ábra).



11. ábra: CIMOM hozzáadása

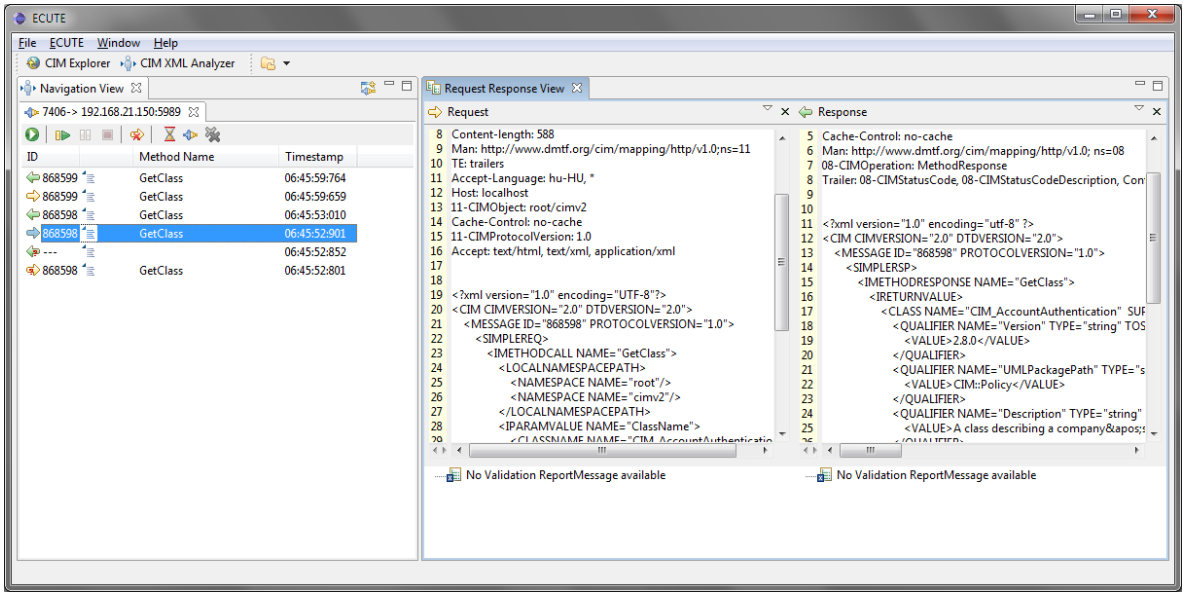


12. ábra: Lekérdezés az ECUTE-ban



13. 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.



14. 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.

7.2 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[19] 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
```

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.

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

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.

7.3 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
```

7.4 További WinRM részletek

A WinRM 2.0-s verziójában nem volt még olyan paraméter, aminek a segítségével az inicializáláskor a hálózati profil típusának ellenőrzését le lehetett volna tiltani.

Ilyen esetekben a Set-WSManQuickConfig által elvégzett feladatokat kézzel kellett elvégezni:

```
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
```

Van még a WS-Man és WinRM megvalósításban egy kevésbé használt funkció, a töredékek kezelése.

1. 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).

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.
```