LDAP PROTOKOLL ALAPÚ WEB FELÜLETŰ MONITOR RENDSZER KIFEJLESZTÉSE

Völgyesi Péter

Budapesti Műszaki és
Gazdaságtudományi Egyetem

2000

Konzulensek:

dr. Miskolczi János (MATÁV-PKI)
dr. Ziegler Gábor (BME-TTT)

TARTALOMJEGYZÉK

ÁBRAJEGYZÉK

1. Fejezet

1. BEVEZETÉS

A legutóbbi évtizedekben jelentkezett információtárolási, -továbbítási, feldolgozási igények jelentős növekedése az alkalmazott eszközpark és szoftverek robbanásszerű gyarapodásához vezetett az élet szinte minden területén. Lassan a feledés homályába merülnek azok az idők, amikor egy vállalat, intézmény megtehette, hogy kizárólag egyetlen informatikai cég termékeit használja belső számítástechnikai igényeinek kielégítésére. A számszerű változás mellett tehát minőségi változások is végbementek ill. ma is tartanak ezen a területen. Nem véletlen, hogy ezen különböző rendszerek egymáshoz történő idomítása: a rendszerintegráció, egyre nagyobb hangsúlyt kap az informatikai projektek során.

A rendszerek, eszközök közötti kommunikáció megteremtése mellett egyre fontosabb olyan felügyeleti megoldások ill. architektúrák kialakítása, melyekkel egy heterogén és sok elemből álló rendszert (pl. IP hálózat, WWW szerver-farm) egységes felületen – a lényeges események kiszűrésével – vagyunk képesek átlátni és irányítani. Ezen a téren a számítástechnika világában meglehetősen ad-hoc megoldások születtek korábban, és a mai eszközökkel elérhető funkcionalitás sem nevezhető minden igényt kielégítőnek. Ennek oka feltehetően a tervezésben ill. a tervezés során „kifelejtett" menedzsment szempontokban keresendő (pl. IP protokoll-család).

Ez a dolgozat alapvetően hálózat-felügyeleti megoldásokkal kíván foglalkozni. A felügyelet kifejezést elsősorban a monitoring jellegű megfigyelésre, esetleges riasztásokra fogom használni, míg a menedzsment szót a beavatkozásra is képes megoldásokra alkalmazom. A megfigyelés és hibadetektálás az egyik legfontosabb és alapvető feladat ezen a téren, ezért helyeztem erre a hangsúlyt.

A dolgozat elején áttekintjük azokat a klasszikus technológiákat, protokollokat, melyeket a hálózat-felügyelet terén alkalmazni lehet, majd egy konkrét menedzsment rendszer felépítésével ismerkedünk meg. A legfontosabb – és általánosnak nevezhető – vonások mellett megpróbálom megvilágítani ezen architektúrák hibáit is, a következő fejezetek előkészítéseképpen.

A következő részekben néhány olyan új paradigmát szeretnék bemutatni, melyek egy részét napjainkban kezdik alkalmazni ezen a téren, vagy meglátásom szerint alkalmas lenne a felügyeleti rendszerek hatékonyabbá tételére.

Elképzeléseim igazolására egy általam készített kísérleti implementációt fogok bemutatni a tervezési fázistól egészen a működtetés során levont következtetésekig. A megvalósítás aprólékos ismertetése helyett a lényeges vagy érdekesnek tűnő vonatkozásokra fogok hangsúlyt fektetni.

Igyekeztem a dolgozat során minél általánosabb megfogalmazásokat, kifejezéseket használni (pl. rendszerfelügyelet, rendszermenedzsment). A gyakorlatban a legelterjedtebb eszközök azonban tagadhatatlanul a TCP/IP alapú hálózatmenedzsmenttel kapcsolatos megoldások. Ennek hatása alól nem mindenhol tudtam kivonni magam. A most következő részben megpróbálom kategóriákba szervezni és pontosabban definiálni a menedzsmenttel kapcsolatos feladatokat, hogy később megengedhessem magamnak ezen fogalmak kicsit felületesebb alkalmazását.

2. Fejezet

2. A RENDSZERMENEDZSMENT FELADATAI, FEJLŐDÉSE

A bevezetőben szó volt a rendszermenedzsment (vagy kontroll) és a felügyelet fogalmak közötti különbségről. Most egy másik alapvető szempont alapján fogjuk osztályozni a feladatokat, nevezetesen a megoldandó probléma vagy funkció típusa szerint **[Sta_99]**. Ezen területek:

- Hibamenedzsment *(Fault Management)*
- Konfiguráció menedzsment *(Configuration Management)*
- Elszámolás menedzsment *(Accounting Management)*
- Teljesítménymenedzsment *(Performance Management)*
- Biztonságmenedzsment *(Security Management)*

Az egyes feladatkörök egy része felügyeleti és kontroll lehetőségeket is tartalmaz, míg mások csak az egyik vagy másik területtel foglalkoznak. Az öt fogalom és a feladatkör pontosabb feltérképezése és megértése segítséget nyújt a menedzsmenttel kapcsolatos eszközök használatához, és elengedhetetlenül szükséges új felügyeleti eszközök tervezése során.

2.1. Hibamenedzsment

Ezzel a területtel fogunk legtöbbet foglalkozni, és a később bemutatásra kerülő megvalósítás is ezt a feladatcsoportot célozza meg. Az egyik legalapvetőbb szolgáltatása a felügyeleti rendszereknek a hibák érzékelése,

naplózása, a hiba okának felderítése, riasztás küldése külső rendszerek felé, ill. „haladó szinten" a hiba elhárításának megkísérlése (ami érintheti egyes eszközök konfigurációjának megváltoztatását). A feladat alapvetően monitor jellegű tevékenységet igényel, a beavatkozás a legtöbb esetben nem nélkülözheti az emberi beavatkozást.

A felhasználók a legtöbb esetben elviselik a szolgáltatások alkalmankénti kimaradását. Az ő elvárásaik ezekben az esetekben a gyors reakció (ami a hiba kijavítására, esetleg áthidalására irányul), valamint a tájékoztatás. Ez utóbbi talán a legfontosabb. A várható kimaradásokról (pl. karbantartási munkák) előzetes értesítést érdemes küldeni, a véletlenszerűen keletkező hibákról utólag ill. már a hiba detektálása során hasznos tájékoztatást adni. Ennek legmegfelelőbb formája a levelezési listák segítségével megvalósított információs *(ticket)* rendszer.

A hibák detektálását, ill. a hibaok megtalálását több tényező nehezítheti a gyakorlatban:

- Lokálisan nehezen felfedezhető hibák (pl. *deadlock* helyzetek)

- Hálózati topológiából adódó függőségek miatti hiba-meghatározás (pl. alacsonyabb protokoll rétegben keletkező hibák befolyásolják a felsőbb szintek működését is)

- Alkalmazások gazda számítógépe okozta bizonytalanság (pl. adott hálózati szegmens megszakadása a rajta levő gépek szolgáltatásainak elérhetőségére is kihat)

- Lokális beavatkozások esetenként megzavarhatják a hiba pontos meghatározását a menedzsment rendszer számára (pl. helyi rendszergazda újraindítja a kiszolgálót, miközben a központi felügyeleti szoftver megpróbálja behatárolni a hibát)

- Sok esetben a hiba egzakt definiálása is problémát jelenthet, mely az üzemeltető ill. a rendszert tervező informatikusok hatáskörébe tartozik (pl. küszöbértékek meghatározása)

A hibamenedzsmenttel kapcsolatos fontos követelmény az említett függőségek miatti csoportos riasztások alapján a lényeg ill. a valódi ok megállapítása. Az üzemeltetés felé minél tömörebb formában kell az eseményt jelezni. Ezzel a területtel részletesebben fogunk még foglalkozni a korrelációvizsgálat címszó alatt.

2.2. Konfiguráció menedzsment

Ez a feladatcsoport a menedzselt rendszer egyes komponenseinek egységes (felületen történő) beállításaival foglalkozik. Nyilvánvalóan ezen funkciók beavatkozó jellegűek, vagyis a kontroll jellegű csoportba sorolhatók. A konfiguráció menedzsment foglalkozik olyan alapvető feladatokkal is, mint az eszközök elindítása, leállítása, újraindítása, az egyes szoftverek frissítése (pl. router operációs rendszere). Az inicializációs tevékenység utáni feladatok az eszközök konfigurációs beállításának követése (pl. ATM kapcsolóban létrehozott PVC-k) és azok fenntartása (pl. újraprogramozással).

A felhasználó elvárása a konfigurációmenedzsmenttel szemben, hogy az egyes beavatkozások ne befolyásolják a szolgáltatások elérését, ill. a jelentősebb változásokról megfelelő tájékoztatást kapjon (információs (ticket) rendszer).

A konfiguráció menedzsmenttel kapcsolatos egyik legnagyobb kihívás a konfigurációs információk (attribútumok) struktúrájának kialakítása, s ezen belül az egyes paraméterek közötti függőségi viszonyok ábrázolása. Néhány lehetséges megközelítés:

- Egyszerű (nevesített) változók listája, melyek korlátozott mértékben struktúrába szervezhetők (pl. tömbök formájában). Ezt a

megközelítést alkalmazza a később ismertetésre kerülő SNMP protokoll.

- Objektum orientált adatbázis, mely a jól elválasztható konfigurációs elemeket objektumokba zárja, és a beállítások ezen objektumok attribútumaiként jelentkeznek. Az *OO* világban megszokott hozzáférési, öröklődési lehetőségek itt is alkalmazhatók. Az OSI világa ezt a megközelítést alkalmazza

- Relációs adatbázisok alkalmazása

2.3. Elszámolás menedzsment

Az elszámolás menedzsment feladata a különböző erőforrások felhasználásának naplózása ill. számszerűsítése. A számlázás során fontos meghatározni azokat a csoportokat, személyeket, projekteket, melyek az erőforrás használata során együtt kezelendők. Sok esetben nem kis problémát vet fel a felhasználó pontos meghatározása az elszámolásért felelős szoftvernek, ilyenkor nincs más megoldás, mint autentikációs mechanizmussal kiegészíteni az erőforráshoz történő hozzáférést (pl. Web proxy felhasználók).

Az elszámolási rendszer nem feltétlenül a költségek felhasználókra terhelését szolgálja. Segítséget nyújthat a szűkös erőforrások (pl. hálózati sávszélesség) igazságosabb vagy célszerűbb felhasználásához, s így közvetve a felhasználók munkáját könnyítheti meg. E téren átfedés jelentkezik a hamarosan bemutatásra kerülő teljesítménymenedzsment témakörrel. Az erőforrások igénybevételének naplózása ill. az ezek alapján készített kimutatások segítenek a további bővítési irányok kijelölésében. Az elszámolás menedzsment kifejezetten megfigyelés jellegű tevékenység, beavatkozásra csak kivételes esetekben használják (pl. szigorú kvóta rendszer érvényre juttatása).

A felhasználók elvárása egyértelmű ezen a téren: korrekt és precíz nyilvántartás, és bármikor lekérdezhető, elérhető (adott felhasználóra vonatkozó) adatok.

2.4. Teljesítménymenedzsment

Szintén a megfigyeléssel kapcsolatos feladatok közé tartozik, mely kritikus esetekben beavatkozást is kiválthat. A teljesítménymenedzsmentet foglalkoztató kérdések:

- Az erőforrások kihasználtságának mértéke *(utilization)*
- A terhelések eloszlása, csomók kialakulása *(burst-ök)*
- A szűk keresztmetszetek meghatározása *(bottleneck-ek)*
- A szolgáltatások igénybevétele során mérhető teljesítménymutatók elfogadhatóak-e: áteresztőképesség, válaszidő, hibaarány, igényvesztés

A felhasználót többnyire az utolsó pont érinti. Ezen paraméterek értékére is általános ill. „legrosszabb eset"-ben kíváncsi. Az első pontok az üzemeltetés számára szolgálnak fontos információkkal a fejlesztéseket, átcsoportosításokat érintő döntések meghozatalához.

A teljesítménymenedzsmenttel kapcsolatos problémák:

- Túl sok, nehezen átlátható adat, melyek közötti kapcsolatok feltérképezetlenek
- Bizonyos indikátorok pontos jelentése nem megfelelően vagy félre értelmezett (akár a gyártó oldalán)
- Néhány teljesítménymutatót csak bizonyos gyártó eszközei képesek produkálni, így a különböző eszközök nehezen összevethetők

- Az eredmények pontos kiértékelése sok időt vesz igénybe, és a kapott eredmény alapján nehéz a konkrét (automatikus) lépések meghatározása

A legfontosabb indikátorokat és rövid magyarázatukat tartalmazza az 1. táblázat.

Szolgáltatás orientált parméterek	
Rendelkezésre állás	Az idő hány százalékában áll a felhasználó rendelkezésére az adott szolgáltatás (pl. esetleges hibák miatti kimaradások)
Válaszidő	Mennyire érzékeny a rendszer, milyen gyorsan válaszol a felhasználói beavatkozásokra
Hibamentesség	A hibás átvitel vagy kiszolgálás százalékos aránya
Kihasználtság orientált paraméterek	
Áteresztő képesség	Az erőforrás időegység alatt hány igényt képes kiszolgálni
Kihasználtság	Az erőforrás elméleti és aktuálisan kihasznált kapacitása közötti százalékos arány

1. táblázat Teljesítményparaméterek

A teljesítménymenedzsmentet megvalósító eszköz három komponensből épül fel: a teljesítmény paraméterek *mérése* az egyes indikátorok aktuális értékének gyűjtését és naplózását jelenti, a teljesítmény *analízis* ezen adatok közötti kapcsolatok felismerésével és a következtetések levonásával foglalkozik; a harmadik feladat egyéni terhelési jellemzők mesterséges előállítása: forgalom *szintézis*, mely segítségével egyedi helyzetek vizsgálhatók ill. reprodukálhatók.

2.5. Biztonságmenedzsment

Az erőforrásokhoz történő hozzáférés szabályozásán túl egyre nagyobb elvárás a biztonságos kommunikációhoz, adattároláshoz szükséges kulcsok (és hozzájuk tartozó digitális igazolások) karbantartása. Ennek megfelelően a biztonságmenedzsment hitelesítési (authentication), hozzáférés-szabályozási (authorization), titkosítási (encryption) és kulcsmenedzsmenttel kapcsolatos (PKI) funkciókkal bír. Nem szabad elfeledkezni a naplózási feladatokról sem, mely az esetleges biztonsági hiányosságok vagy támadási kísérletek

felderítésében kap nagy hangsúlyt. A biztonságmenedzsment a fentiek alapján mind a monitoring jellegű, mind a kontroll jellegű tevékenységekhez besorolható.

A felhasználók elvárása a biztonsági funkciók terén a lehető legfájdalommentesebb procedúrák használata (pl. belépés), a minél homogénebb integrált rendszerek (közös jelszóadatbázisok, single sign on lehetőségek), a személyes információk védelme *(privacy)*, és a felhasználóra vonatkozó információk (pl. biztonsági naplók), szabályok elérhetősége. Ezen kívánalmak legkönnyebben egy írásban rögzített biztonsági politikával *(security policy)* elégíthetők ki, mely mind a felhasználók, mind az üzemeltetés munkáját egyszerűbbé teszi.

A biztonsággal kapcsolatos feladatok és technológiák már most előkelő helyet foglalnak el az informatika világában, és ezek jelentősége a jövőben is rohamosan növekedni fog. A terület nagysága miatt ebben a dolgozatban csak érintőlegesen fogunk foglalkozni ezekkel a vonatkozásokkal.

2.6. A menedzsment rendszerek fejlődése

A TCP/IP alapú rendszerek rohamos terjedésével az utóbbi években (évtizedben) egyre alapvetőbb szükségletként jelentkezik ezen hálózatok karbantartását, működtetését támogató menedzsment rendszer.

Az első ilyen feladatot ellátó hálózati protokoll az *ICMP* volt (amit a népszerű *ping* parancs is használ). Ennek segítségével már triviális karbantartási ill. hibadetektálási feladatokat meg lehetett oldani.

A hálózatok egyre bonyolultabbá válásával azonban szükség volt egy olyan protokollra, mely az elérhetőségen kívül jóval több információval tud szolgálni egy hálózati elemről ill. beavatkozhat az eszköz működésébe. A TCP/IP hálózatok világában az *SNMP (Simple Network Management Protocol)*, míg az OSI családban a *CMIP (Common Management Information Protocol)* architektúrát

fejlesztették ki erre a célra. Bár a két protokoll ill. szemléletmód jelentősen különbözik, a menedzsmentet érintő objektumok hierarchiája *(MIB, Management Information Base)* a két rendszerben megegyezik. Ez főként azért volt fontos, hogy egy esetleges átállás a használatban levő SNMP-ről a jóval robosztusabb CMIP-re zökkenőmentesebb lehessen. Egyelőre úgy tűnik azonban, hogy az SNMP még jó ideig megmarad az elsődleges platformként. (Új verziók kidolgozásával próbálják a hiányzó funkciókat – pl. biztonság – pótolni.)

A fenti protokollok önmagukban nem alkalmasak komplex rendszerek kezelésére, ezt a feladatot az ezekre épülő hálózatmenedzsment alkalmazások látják el. Számos kisebb-nagyobb ilyen alkalmazás létezik ma, néhány jelentősebb ezek közül a HP OpenView, Tivoli, stb. Ezek az alkalmazások általában egy mag köré épülő kisebb eszközökből (tools) állnak. A hálózatmenedzsment egyik legnagyobb problémája a hálózati eszközök sokfélesége ill. az általuk szolgáltatott információk gyártóspecifikus volta (nem is beszélve a nem SNMP felületen keresztül menedzselhető eszközökről). Ezek miatt alapvető követelmény ezekkel az alkalmazásokkal szemben, hogy könnyen lehessen integrálni saját (ill. a gyártó által készített) szoftvermodulokat.

Az SNMP ill. CMIP protokollokkal fogunk részletesebben megismerkedni a következő fejezetben, majd a menedzsment alkalmazások tipikus szerkezetével és funkcióival fogunk foglalkozni.

3. Fejezet

3. FELÜGYELETI PROTOKOLLOK ÉS SZABVÁNYOK

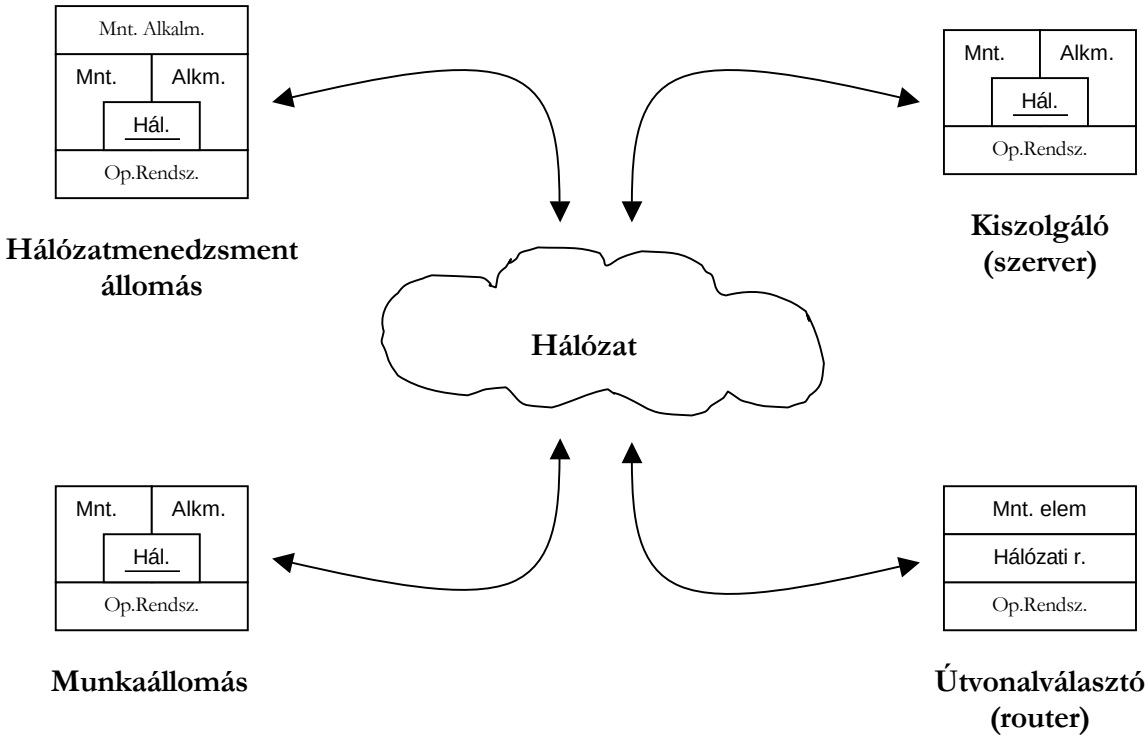
Ebben a fejezetben megismerkedünk a menedzsment rendszerek általános szerkezetével, a felhasznált protokollokkal és egyéb technológiákkal. Ahogy a bevezetőben is felhívtam rá a figyelmet, egyre nagyobb hangsúlyt fog kapni a hálózatmenedzsment, ill. a hibadetektálás.

Olyan építőkövekkel ismerkedünk tehát meg, melyek a mai menedzsment rendszerek legfontosabb alapjai. Elsősorban a TCP/IP alapú hálózatokban elterjedt SNMP protokollról és a hozzá kapcsolódó fogalmakról lesz szó, de említést kap az OSI világ CMIP menedzsmentje is. A tendenciák azt mutatják, hogy a CMIP elterjedésével rövidtávon (és feltehetően középtávon) nem lehet számolni a jelenlegi hálózatokon. Ezt támasztja alá W. Stallings könyvének 3. kiadása is **[Sta_99]**, ahol a CMIP protokoll kiszorult a címből és a fejezetekből. Megemlítése mégis fontos, hogy jobban értsük az SNMP hiányosságait ill. más lehetőségeket.

3.1. Menedzsment rendszerek általános szerkezete

A hálózatmenedzsment rendszer a felügyeleti és szabályozási feladatokat ellátó eszközök halmazát ill. integrációját jelenti. Fontos elvárás a menedzsment rendszerrel szemben, hogy egységes felületen keresztül biztosítsa az üzemeltetési feladatok elvégzését.

Az 1. ábra egy tipikus menedzselt hálózatot mutat be. Minden menedzselt eszköz tartalmaz olyan modult vagy modulokat, melyek a távoli lekérdezést ill. beavatkozást segítik. A modulokat sokszor ágenseknek (agent) hívjuk.



1. ábra A menedzsment architektúra elemei

[Mnt. – menedzsment protokoll, Hál. – hálózati támogatás, Alk. – alkalmazás, Mnt. Alk. – központi menedzsment program]

Ezek a menedzsment elemek a következő feladatokat látják el:

- Statisztikai adatok gyűjtése az eszköz ill. erőforrás igénybevételéről, hibáiról, valamint ezek lokális tárolása.
- Válasz a központi felügyeleti állomás kéréseire:
- Kritikus esetekben riasztás küldése a központ felé

A menedzselt erőforrásokon futó ágens legfontosabb feladata, hogy az adott eszköz hardver és szoftver specifikus információit egy közös nyelvre – az alkalmazott hálózatmenedzsment protokollra – fordítsa.

A menedzsment központ futtatja a menedzsment alkalmazást, mely a legtöbb esetben egy áttekintő és hierarchikus hálózati térképből áll, melyen elhelyezett objektumokon különböző műveleteket kezdeményezhetünk.

A menedzsment alkalmazás alapvetően három rétegből áll:

- Felhasználói felület biztosítása, megjelenítés: az egységes kezelőfelület kialakítása, áttekinthetőség ill. különböző nézetek (view) biztosítása

- Menedzsment információk értelmezése: a statisztikai és egyéb adatok begyűjtése, értelmezése – mely sokszor eszköz ill. gyártó specifikus modulokat igényel, a felhasználói felületről kezdeményezett műveletek „kivitelezése".

- Kommunikációs protokollok és adatbázis támogatás: különböző menedzsment (pl. SNMP) és transzport (pl. UDP, IPX) protokollok biztosítása a menedzsment réteg számára. A menedzsment információk tárolásához adatbázis illesztők biztosítása (pl. ODBC meghajtók)

A megbízhatóság növelése érdekében sok esetben redundáns felügyeleti munkaállomásokat helyeznek üzembe. Egyszerre egy ilyen munkaállomáson dolgoznak az üzemeltetők, a többi állomás csak adatokat gyűjt, ill. gondoskodik, hogy konfigurációja, topológiai információja szinkronban maradjon az elsődleges géppel. Hiba esetén így zökkenő mentesebb az átállás. A szinkronizációt sokszor nagyon nehéz tökéletesen megoldani, és a tartalék állomások által generált többletforgalom is problémákat okozhat.

3.2. Lekérdezések és riasztások

A hálózati problémák észlelése alapvetően kétféleképpen történhet. Az első megközelítés szerint a központi felügyeleti állomás a megfigyelt objektumokat, erőforrásokat folyamatosan ellenőrzi, adott idő intervallumonként lekérdezi,

teszteli az eszközöket. A hiba detektálása a rossz vagy hiányzó válasz alapján történik.

A másik – és haladóbb – megközelítés szerint, az eszközök saját diagnosztikai képességekkel rendelkeznek, mely lehetővé teszi, hogy a berendezés észlelje a működés során jelentkező hibákat. Az esetleges hibákról riasztást küld a központ felé. A központ a „bejelentés" után pontosabb információkhoz juthat további lekérdezések segítségével.

A folyamatos lekérdezés legnagyobb problémája, hogy nagy megfigyelt objektumhalmaz esetén jelentős forgalmat generálhat a hálózaton ill. a menedzsment munkaállomással jóval nagyobb erőforrásigényt támaszt. A mérés tehát sokszor befolyásolja a mérendő rendszert ill. az eredményeket. (Pl. kis sávszélességű WAN kapcsolat terheltségének gyakori folyamatos lekérdezése a kapcsolaton keresztül).

A riasztás alapú megközelítés sem tökéletes. Teljes kieséseket – ahol a diagnosztikai modul is meghibásodik – nehéz pusztán ezzel a módszerrel észre venni. Minden hálózati eszközön pontosan be kell állítanunk a menedzsment állomás címét (és esetleges költözéskor ezt szintén minden berendezésen meg kell változtatnunk.) A ma használt riasztási módszerek nyugtázatlanul működnek (vagyis a menedzsment állomás nem küld nyugtacsomagot a riasztást generáló eszköznek), így a riasztás biztos célba jutása nem biztosítható.

A folyamatos lekérdezés során érdemes megvizsgálni a skálázhatóság határait. Egyszerűsítésképpen feltételezzük, hogy a menedzsment állomás egyszerre egy ágenssel foglalkozik (vagyis, amíg nem érkezik válasz, addig nem küld kérést a következő ágens felé). A következő paraméterek befolyásolják a monitoring korlátait:

- Hálózati eszközök (ágensek) száma (N)

- Átlagos válaszidő (függ a lekérdezett információ mennyiségétől, az ágens teljesítményétől és a hálózati késleltetéstől) (Δ)

- Lekérdezési frekvencia ill. két lekérdezés között eltelt idő (gyakorlati alkalmazásoknál ez 5-15 perc között változik) (T)

E három mennyiség között a következő nyilvánvaló összefüggés áll fenn:

$$N \leq \frac{T}{\Delta}$$

Mit jelent ez a gyakorlatban? A BME hálózatán végzett kísérletek alapján egy ilyen típusú LAN hálózaton egy SNMP változó teljes lekérdezési ideje kb. 0.25-0.5 másodperc. 10 perces lekérdezési időközöket feltételezve tehát már így is egy 1200-2400 közötti felső korlátot kapunk. Bár a lekérdezések átlapolásával ez a korlát kitolható, a skálázhatóság határai azért jól látszanak.

3.3. Elosztott felügyeleti megoldások, proxy-k

A ma használt legtöbb menedzsment megoldás a központi menedzsment munkaállomás koncepcióra épül. Ennek nyilvánvaló hátrányai:

- Egy helyről generált túl nagy hálózati forgalom

- Kritikus pont az architektúrában (single point of failure)

- A menedzsment feladatok delegálása nagyon nehezen vagy egyáltalán nem oldható meg.

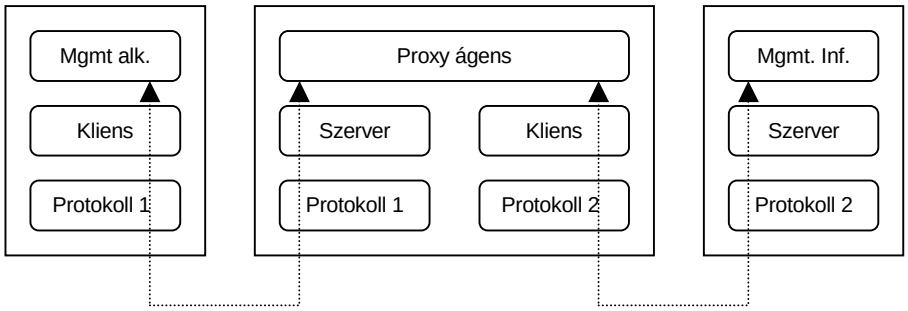
- Skálázhatóságai problémák (pl. menedzsment állomás erőforrásai)

A fentiek miatt egy valóban komoly megoldásnak biztosítania kell az elosztott működés lehetőségét. Ez alapvetően két megközelítéssel történhet.

Az egyik szerint a menedzsmentalkalmazást több munkaállomásra telepítik, majd ezek között hierarchikus viszonyokat alakítanak ki (ezt a lehetőséget az adott alkalmazásnak támogatnia kell). Az egyes állomások valamilyen belső protokollon keresztül kommunikálnak egymással. A legfelső szintű állomásnak teljes kontrollja van a hálózat felett, azonban az információk az egyes tartományokból már jelentős mértékben szűrve és értelmezve érkeznek hozzá. Lokális – csak az adott tartománybeli eszközöket érintő - feladatokat a alacsonyabb munkaállomásokon is el lehet végezni.

A másik megközelítés szerint egyes tartományokba ún. proxy ágenseket helyeznek el. Ezek olyan modulok (szoftver, hardver elemek), melyek az adott tartomány eszközeivel kommunikálnak, azokról adatokat gyűjtenek, és eközben az így szerzett információkat egy külső menedzsment állomás számára elérhetővé teszik. Ebben az esetben a központi állomás ezekkel a gyűjtő ágensekkel beszélget szabványos menedzsment protokollokat használva. Egy nagyon látványos gyakorlati alkalmazása ennek a Cisco WAN Manager nevű termék, mely egy komplex ATM hálózatot egyetlen proxy-n keresztül mutat, miközben a proxy a hálózat ATM kapcsolóival beszélget, ill. útvonalválasztási, erőforrásallokációs feladatokat lát el.

A proxy ágensek másik nagy alkalmazási területe az olyan eszközök kezelése, melyek közvetlenül nem képesek az alkalmazott menedzsment protokoll használatára. Ezekben az esetekben a proxy ágensnek lehetősége van eszközfüggő módón (pl. RPC, TELNET, TFTP) megszólítani ezeket az elemeket. Így tulajdonképpen átjárót biztosítanak különböző hálózatmenedzsment protokollok között. Egy ilyen proxy kialakítását mutatja a 2. ábra.



2. ábra Proxy ágens használata

3.4. A megfigyelt és kezelt változók

Mind az SNMP mind a CMIP hálózatmenedzsment a kezelt eszközöket, erőforrásokat változókon keresztül látja, ill. vezérli. Ezen változók egy (virtuális) adatbázisban vannak tárolva, melyet menedzsment információs adatbázisnak (management information base – MIB) hívnak. A virtuális szó itt arra utal, hogy a kezelt adatok nem egy konkrét és fizikailag összefüggő adatbázisban helyezkednek el, hanem igen sokféle forrásból állítja elő a kihelyezett ágens, a külvilág – és így a menedzsment állomás – felé viszont már koherens és rendezett képet mutat.

Hosszú időn keresztül törekedtek arra, hogy a CMIP és az SNMP világban megőrizzék a közös információs bázist, vagyis mindkét protokoll azonos változókat, azonos struktúrában kezeljen. Ezzel tulajdonképpen az volt a szabványosítási eljárásban részt vevők szándéka, hogy a későbbi SNMP → CMIP migráció egyszerűbben történhessen. Ez a törekvés azonban egy idő után nem volt tartható, és jelentőségét csökkentette az a tény, hogy az SNMP egyre kevésbé tűnt átmeneti megoldásnak. Így ma – bár sok közös fogalommal operál a két rendszer – a kezelt objektumok szerkezete már nem azonos.

Az SNMP valóban csak egyszerű változókat – skalárokat – kezel, melyek különféle típusúak lehetnek. A bonyolultabb struktúrákat már alkalmazás szinten (pl. a menedzsment állomás moduljaiban) értelmezzük, a protokoll nem foglalkozik ezek kezelésével (ellentétben a CMIP rendszerrel). Az SNMP a változókat fa gráfba szervezi, ahol az egyes értékek csak a fa leveleiben találhatók. Adott változót egyértelműen kijelöli a fában megadott elérési útja, mivel az elágazási pontoknál a gyermekeket egyértelműen címkével és számmal látjuk el. A közös menedzsment protokoll mellett két területet kellett egységessé tenni, hogy az egységes menedzselhetőség alapvetően biztosított legyen:

1. Ki kellett alakítani azokat az alaptípusokat, melyeket a változók definiálása során felhasználhatunk. Meg kellett határozni ezek kódolását a hálózati átvitelhez. Szabványosították azt az eljárásokat, makrókat, melyekkel később az egyes menedzsment változókat definiálni lehet. Ezeket a szabályokat gyűjtőnéven menedzsment információ struktúrának (structure of management information – SMI) nevezik, és az RFC1155 dokumentum foglalkozik vele. **[RFC1155]**

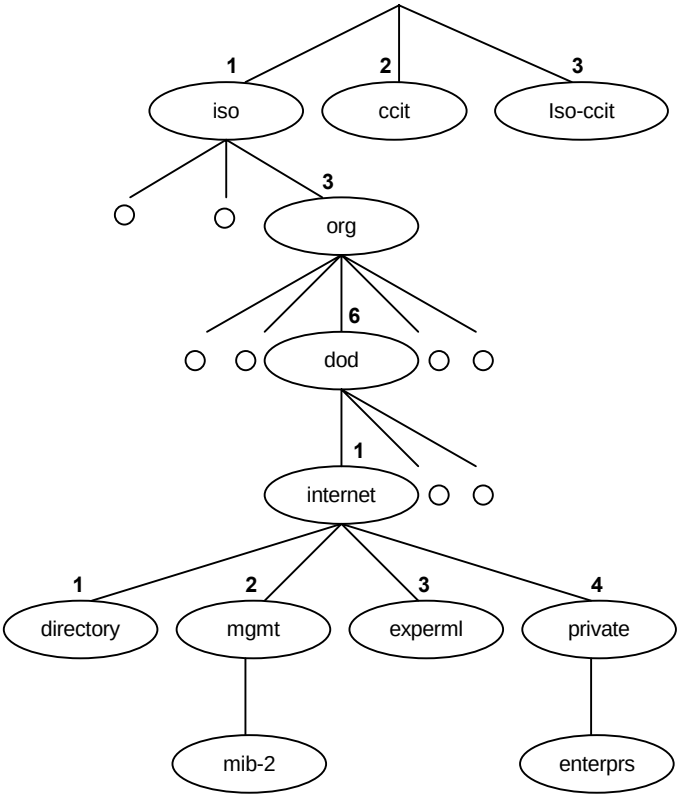
2. Definiálták azokat az alapvető változócsoportokat, melyek az SNMP menedzselhető eszközöknek meg kell valósítaniuk, ill. ha megvalósítanak, akkor azt hogyan kell megtenniük **[RFC1213]** . Ezeket a csoportokat – mint sok más hálózatmenedzsmenttel kapcsolatos fogalmat – ASN.1 nyelven definiálták, és felhasználták az 1. pontban leírt keretrendszert. A csoportokat a gyakorlatban szintén MIB-eknek nevezik. Néha zavaró lehet, hogy a MIB kifejezést a teljes fára vagy csak egy részfára alkalmazzuk.

Az ASN.1 nyelven készített MIB definíciók szöveges file-okban találhatók, és minden komolyabb menedzsment alkalmazás képes ezen leírások importálására, értelmezésére. Ezzel lehetőséget teremt, hogy a gyártó saját

kiegészítései, többlet funkciói is elérhetővé váljanak a menedzsment állomásról. Az importálás azonban nem jelenti azt, hogy a menedzsment rendszer képes az új MIB-et ill. a benne található információt teljes mértékben kezelni, ilyenkor csak szintaktikailag készítjük fel az új részfa befogadására, a szemantikai értelmezéshez nem ad segítséget az ASN.1 modul (csak emberi fogyasztásra alkalmas megjegyzések, magyarázatok szintjén).

Mit nyerünk mégis a MIB leírás gépi értelmezésével? Egyrészt képesek leszünk a MIB-ben szereplő változók névvel történő elérésére, másrészt bizonyos esetekben (pl. változó módosítása) fontos tudnunk, hogy az adott MIB változó milyen típusú ill. milyen műveletek végezhetők el rajta. Az ágens oldalán tulajdonképpen nincs szükség a MIB definíciós file-ra, hiszen ott maga a szoftvermodul kódja tartalmazza implicit módon a MIB leírást.

A menedzselt változók fa gráfjának alap szerkezetét szintén szabványosították. Az alapszerkezet nagyon fontos ahhoz, hogy a különböző gyártók kiegészítései ne fedjék át egymást, ill. ne ütközzenek későbbi közös szabványokkal. Az alapszerkezetet mutatja a 3. ábra.



3. ábra Az alap OID szerkezet

Az SMI dokumentum a dod csomópont alatt elhelyezett internet címkéjű pont alá helyezi az összes menedzsment objektumot. Ennek a pontnak az ASN.1 definíciója:

```
internet OBJECT IDENTIFIER ::= {iso(1) org(3) dod(6) 1}
```

Az *internet* címke alatt négy nagy csoport található:

- *directory:* jelenleg nem használt, további fejlesztésekre fenntartott
- *mgmt*: az IETF által RFC-kben szabványosított alapvető MIB-ek ez alatt helyezkednek el. Ezen belül is a mib-2 az a hely, ami alatt a standard változókat találjuk

- *experimental*: kísérleti, átmeneti jelleggel ez alatt bárki létrehozhat MIB részfákat. Ezeket éles rendszerben nem szabad használni, hiszen az átfedések nincsenek kiküszöbölve.

- *private*: gyártók által kifejlesztett kiegészítések, gyártófüggő megoldások MIB gyökere. Egyetlen gyermeke, az *enterprises* alatt minden gyártók kérhet magának egy csomópontot (névvel és a hozzá tartozó számmal), mely alá saját MIB-jeit ültetheti.

Szó esett már arról, hogy a MIB definíciókat ASN.1 nyelven (szintakszissal) állítjuk elő. Ez egy olyan formális leíró nyelv, melyet a CCITT és az ISO karöltve fejlesztett ki absztrakt adattípusok, struktúrák, értékek definiálásához. Ezen a téren az ASN.1 széles körben elfogadottá vált. Alkalmazási adatstruktúrák leírásától protokoll csomagok (PDU-k) szerkezetéig nagyon sok feladatra alkalmas.

Fontos megkülönbözetni az adatszerkezetek leírása során (legalább) két szintet. A felsőbb szint az adatok *absztrakt szintakszisával* foglalkozik. Két alkalmazási rétegben elhelyezkedő szoftver nem közvetlen módon kommunikál egymással, hanem az alacsonyabb rétegeken keresztül. Ezen magas szintű rétegek kapcsolata tehát csak virtuális. Fontos, hogy ilyenkor is a két fél megegyezzen az adatstruktúrában, melyet használnak, de a konkrét tárolás, fizikai adatreprezentáció jelentősen eltérhet a két oldalon (pl. ASCII és UTF-8). A kommunikációs csatornán egymással közvetlen kapcsolatot teremtő rétegeknek azonban már a fizikai ábrázolásban is meg kell egyezniük, ezt nevezik *transzfer szintakszisnak*. A transzfer és az alkalmazásszintű szerkezet között teremtene kapcsolatot a megjelenítési réteg, mely a legtöbb mai hálózati implementációban nincs elkülönítve, így jobb híján az alkalmazás szintű szoftverek tartalmazzák ezeket a rutinokat is.

Az ASN.1 – ahogy neve is utal rá – az absztrakt szintakszis leírásával foglalkozik. Szerencsére szabványosítottak olyan eljárásokat, melyekkel az így leírt struktúrák bitsorozattá alakíthatók. Ezen kódolási szabvány neve: basic encoding rules (BER). Az SNMP és a CMIP a MIB változókat ill. a protokoll PDU-kat a BER szerint viszi át. A BER kódolás nagy előnye, hogy „öndokumentáló" szerkezetű, ami a gyakorlatban azt jelenti, hogy a kódolás során a bitfolyamba nem csak az egyes mezők típusai, hanem azok adattípusai ill. hossza is bekerül. Tehát a fogadó oldalon anélkül is szét lehet bontani egy ilyen struktúrát, hogy rendelkeznénk az ASN.1 leírásával. Az adattípusok BER kódjai egy – hamarosan ismertetetésre kerülő – osztályazonosítóból és az osztályon belüli kódból áll.

Az ASN.1 nyelv alaptípusokat és azok módosításához szükséges eszközöket biztosít az új adatszerkezetek leírásához. Az alaptípusok az ún. *universal* osztályba tartoznak, és azonosító számmal rendelkeznek (a BER típuskódolás miatt). Amikor új típusokat származtatunk, akkor a következő lehetőségek közül választhatunk:

- strukturált összetett típusok kialakítása (`SEQUENCE, SET` műveletek)
- egyszerű típusok finomítása, definiálása (tagged types)
- egyéb, strukturálatlan típusok (`ANY, CHOICE`)

Az így létrehozott típusok négy kategóriába (osztályba) sorolhatók, amit a típus létrehozásakor választunk ki. Ez az adott típus érvényességi körét és a konkrét BER kódolásnál használt típus azonosító értékét befolyásoja. A négy kategória:

- *universal*: erről már volt szó, általános érvényű típusok, általában nem definiálunk ilyeneket, mivel ez a szabványosító testületek feladata

- *application*: adott alkalmazás (felhasználás) számára definiált típus, csak az adott alkalmazásban van értelmezve, szintén az adott alkalmazást, protokollt szabványosító szervezetek feladata ezek meghatározása

- *context*: adott alkalmazáson belül korlátozott érvényességi körrel rendelkező típus (nem a teljes alkalmazásban értelmezett)

- *private*: felhasználók, gyártók által definiálható típusok osztálya

Az ASN.1 nyelvben definiált *universal* osztályú típusokat tartalmazza a 2. táblázat. Érdemes megjegyezni, hogy a struktúra képző operátorok is típusként vannak definiálva.

TÍPUSKÓD	TÍPUS NEVE	LEÍRÁS
UNIVERSAL 1	BOOLEAN	*Értéke igaz vagy hamis (true vagy false)*
UNIVERSAL 2	INTEGER	*Egész számok*
UNIVERSAL 3	BIT STRING	*Tetszőleges számú bit sorozata*
UNIVERSAL 4	OCTET STRING	*Tetszőleges számú oktet (bitnyolcas) sorozata*
UNIVERSAL 9	REAL	*Valós számok*
UNIVERSAL 10	ENUMERATED	*Egész számok adott (explicit módon felsorolt) halmaza*
UNIVERSAL 6	OBJECT IDENTIFIER	*A korábban bemutatott fa gráf alapú elnevezési konvenció szerinti objektum azonosítók*
UNIVERSAL 7	Object descriptor	*Emberi fogyasztásra alkalmas leírása az adott típusnak, objektumnak*
UNIVERSAL 18	NumericString	*0-9 közötti decimális számjegyek jegyek és szóközök sorozata*
UNIVERSAL 19	PrintableString	*Nyomtatható (képernyő) karakterek*
UNIVERSAL 20	TeletexString	*CCITT T.61 ajánlás által meghatározott karakterhalmaz*
UNIVERSAL 21	VideotexString	*CCIT T.100 és T.101 által meghatározott karakterhalmaz*
UNIVERSAL 22	IA5String	*ASCII karakterkészlet elemei*
UNIVERSAL 25	GraphicString	*ISO8824 karakterkészlet*

UNIVERSAL 26	VisibleString	*ISO 646 karakterkészlet*
UNIVERSAL 27	GeneralString	*Általános (nem definiált kódkészletű) karaktersorozat*
UNIVERSAL 5	NULL	*A klasszikus érték nélküliség (vagy definiálatlanság) jelölésére*
UNIVERSAL 8	EXTERNAL	*Külső dokumentumban értelmezett (az ASN.1-en kívüli universal változó)*
UNIVERSAL 23	UTCTime	*Időpont jelölésére bevezetett típus (másodperc pontossággal, időzóna megjelölésével)*
UNIVERSAL 24	GeneralizedTime	*Az előzővel megegyező, de az évszámot négy jeggyel tároló időpont típus*
UNIVERSAL 9-15	Fenntartott	
UNIVERSAL 28-	Fenntartott	
UNIVERSAL 16	SEQUENCE (OF)	*Adott típusok egymás utáni rendezett sorozata, klasszikus struktúra definiálása*
UNIVERSAL 17	SET (OF)	*Adott típusok halmaza, ahol a sorrend nem értelmezett*

2. táblázat Az ASN.1 univerzális típusai

A MIB leírásokat bár ASN.1 nyelven készítik, mégis csak a nyelv elemeinek egy részhalmaza használható. Ennek oka, hogy teljes értékű ASN.1 értelmező készítése nem könnyű feladat, valamint a MIB leírások sokkal olvashatóbbak, ha egységesen kötött forma szerint készülnek. A megkötések így is lehetővé teszik, hogy a gyakorlatban előforduló menedzselt objektumok MIB változóit leírjuk velük. A szigorítások mellett némi segítség, hogy definiálták a menedzsmenthez elengedhetetlenül szükséges típusokat (application osztály szintjén), ill. egy olyan makrót szabványosítottak, mely segítségével egy MIB változó nagyon könnyen és teljes értékűen leírható.

A legfontosabb SNMP típusok:

- *networkaddress*: Több címcsalád közül választható union jellegű típus. (CHOICE operátorral definiált)

- *ipaddress*: 32 bites cím IP alapú hálózatok elemeinek címzéséhez

- *counter*: nem negatív értékű 32 bites egész, mely monoton növekszik, maximális értékének elérésekor nullázódik és újraindul (pl. hálózati interfész forgalom)

- *gauge*: nem negatív értékű 32 bites egész, mely növekedni és csökkeni is képes. A felső határ elérésekor bennragad (vagyis a maximális érték elérésekor nem változik tovább)

- *timeticks*: nem negatív egész, mely időpontot reprezentál, egy adott referencia időpont (epoch) óta eltelt századmásodpercekkel

- *opaque*: tetszőleges adat, melyet az átvitel során OCTET STRING-ként kódol a rendszer

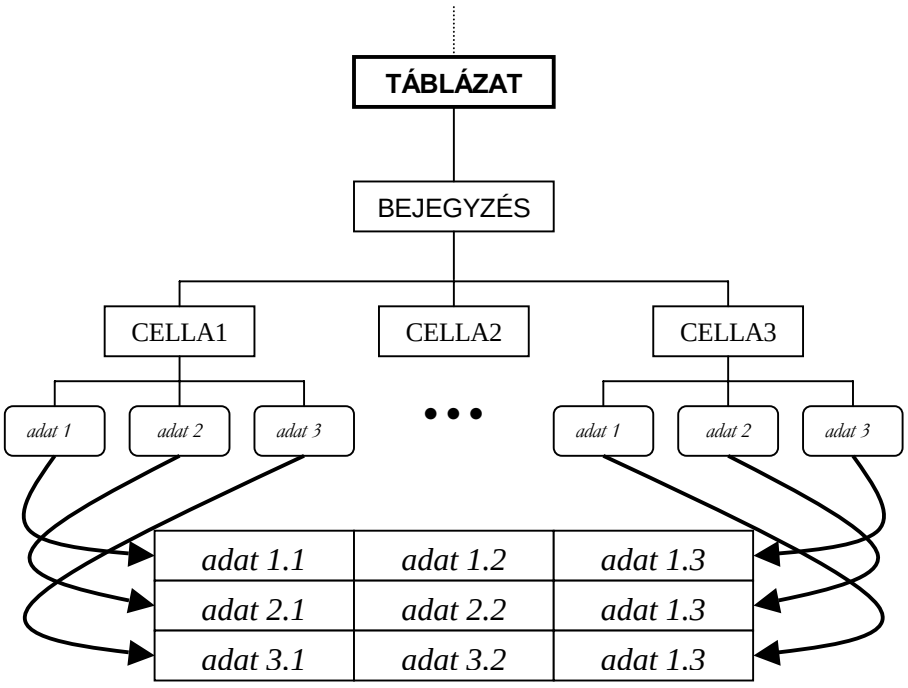
Az SNMP protokoll kizárólag skalár jellegű változókkal foglalkozik. Az SNMP rendszerben (az SMI-t taglaló RCF-ben) azonban a táblázatok ábrázolását lehetővé teszik. Ez sokszor nagyon fontos a hálózatmenedzsment során, gondoljunk csak a hálózati interfészek, lemezes egységek, processzorok statisztikai adataira – ezeket egyedül táblázatos formában célszerű kezelni. Az SNMP-ben a táblázatok definiálása az ASN.1 SEQUENCE és SEQUENCE OF operátorával történik.

A táblázatokat olyan fában (MIB) kell tehát ábrázolni, hogy az egyes cellaértékek a fa levelein jelentkezzenek csak. Ehhez a következő fogalmakat (objektumokat) vezetjük be: tábla (Table), bejegyzés (Entry), cella. A cellák különböző típusúak lehetnek és egy bejegyzést (rekordot) tesznek ki, a

bejegyzések között már nincs típuskülönbség. Hamarosan megismerkedünk egy konkrét MIB leírás részletével, most a könnyebb érthetőség miatt egy egyszerű ASN.1 táblázatleírást mutatok meg:

```
SampleTable ::= SEQUENCE OF SampleEntry
SampleEntry ::= SEQUENCE {SampleCell1 INTEGER,
                          SampleCell2 OCTET STRING,
                          SampleCell3 INTEGER}
```

A MIB fában legalul lesznek az egyes cella értékek, a megfelelő cella alatt az összes oszlop értéke szerepelni fog. A 4. ábra világosabbá teszi a fa ábrázolását.



4. ábra MIB táblázatok szerkezete

A hagyományos ASN.1 nyelvet az SNMP és OSI világban egy nagyon hasznos makróval egészítették ki. Ennek alapgondolata, hogy mindkét menedzsment architektúra menedzselt objektumokkal dolgozik – még ha ezek az SNMP esetén egyszerű skalárok is. A MIB leírás tehát ezeknek az

objektumoknak a definiálására és elnevezésére szolgál. Ésszerűnek tűnik tehát egy olyan makró konstruálása, mely kifejezetten egy menedzselt objektum leírására szolgál. Ezzel nem csak a MIB fejlesztők munkáját egyszerűsítették, de az egységes és mindenki számára könnyen (könnyebben) értelmezhető MIB definíciók irányába is nagyot léptek az SMI megalkotói.

A makró neve OBJECT-TYPE. A makró használata során jön létre egy új MIB objektum, ehhez a következő komponenseket – ASN.1 makró paramétereket - kell megadnunk:

- SYNTAX: az ASN.1 leírása az adott objektum típusának (pl. INTEGER vagy SEQUENCE OF REAL)

- ACCESS: az objektum elérhetőségét (a menedzsment állomás felől) deklarálja (lehetséges értékei SNMP esetén: *read-only, read-write, write-only, not-accessible*)

- STATUS: az objektum megvalósításáról szolgál információval (*mandatory, optional, depeicated, obsoleted*) A deprecated már elavult, de kötelezően megvalósítandó elem, míg az obsoleted állapotú objektumot nem valósítják meg.

- DESCRIPTION: opcionális makró paraméter az objektum szöveges leírására (mint egy megjegyzés)

- REFERENCE: kereszthivatkozás egy másik MIB-ben definiált objektumra (opcionális, és emberi feldolgozásra szolgáló lehetőség)

- INDEX: Táblázatok definiálása esetén a táblázat azon oszlopainak meghatározása, melyek az adott bejegyzést (rekordot) egyedivé teszik.

- DEFVAL: Opcionális paraméter, mely az adott objektum példányosítása során az új egyed kezdeti értékét határozhatja meg.

- Value Notation: Az objektum elnevezése a MIB fában. (rekurzív módon szokott megvalósulni)

A következő példa egyrészt az OBJECT-TYPE makró használatára, másrészt a táblázatok definiálására mutat példát. Az „állatorvosi ló" a hálózati interfészeket leíró MIB (RFC2233) lesz. Ebből az interfésztábla leírásának egy részletét ragadtam ki.

```
ifTable OBJECT-TYPE
    SYNTAX     SEQUENCE OF IfEntry
    ACCESS not-accessible
    STATUS     mandatory
    DESCRIPTION
        "A list of interface entries.  The number of
entries
         is given by the value of ifNumber."
    ::= { interfaces  2 }
```

Első lépésben magát a táblázat objektumot definiáljuk, mely a rekordok szekvenciáit tartalmazza. Fontos látnunk, hogy a táblázat objektum közvetlenül nem elérhető, mivel ez egy „virtuális mankó" a táblázat leírásához. A táblázatot az *interfaces* csomópont alatt a 2. elágazásként vesszük fel.

```
ifEntry OBJECT-TYPE
    SYNTAX     IfEntry
    ACCESS not-accessible
    STATUS     mandatory
    INDEX      { ifIndex }
    ::= { ifTable  1 }
```

A táblázat definiálása után a sorokat reprezentáló bejegyzés objektumot írjuk le. Ez két lépésben történik. Először az OBJECT-TYPE makróval a táblázathoz hasonlóan definiáljuk a virtuális bejegyzés objektumot, melyet a fában a táblázat objektum alatt helyezünk el. A SYNTAX-nál azonban egy speciális típust adunk meg, mely ezt a bejegyzést határozza meg. Figyeljük meg, hogy a típusnév és az objektumnév a szó eleji betűben eltér!

```
IfEntry ::=
    SEQUENCE {
        ifIndex
            InterfaceIndex,
        ifDescr
            DisplayString,
        ifType
            IANAifType,
        ifMtu
            Integer32,
        . . .
}
```

A bejegyzés definiálásának második lépcsője, hogy az imént említett bejegyzés típust meghatározzuk. Ez a típus egy szekvencia lesz, mely az egyes oszlopokhoz tartozó értékeket tartalmazza.

```
ifIndex OBJECT-TYPE
    SYNTAX     InterfaceIndex
    ACCESS read-only
    STATUS     mandatory
    ::= { ifEntry  1 }

ifDescr OBJECT-TYPE
    SYNTAX     DisplayString
    ACCESS read-only
    STATUS     mandatory
    ::= { ifEntry  2 }

ifType OBJECT-TYPE
    SYNTAX     IANAifType
    ACCESS read-only
    STATUS     mandatory
    ::= { ifEntry  3 }

ifMtu OBJECT-TYPE
    SYNTAX     Integer32
    MAX-ACCESS read-only
    STATUS     mandatory
    ::= { ifEntry  4 }
```

A visszamaradó részben a bejegyzés különböző celláit határozzuk meg. Az eredeti leírásnak a fenti példa csak egy kis részét mutatja.

A gyártók és felhasználók kényelmét szolgálja a MIB-ek nagy fokú modularitása, vagyis az a megközelítés, hogy az egyes MIB részfák egymástól függetlenül implemetálhatók. A gyártók ezért termékeik brosúráin fel szokták

tüntetni, hogy az adott eszköz mely szabványos MIB-eket valósítja meg. Sokszor ez nagyon fontos szempont egy hálózati elem kiválasztásakor.

Sajnos, sokszor nem egészen világos, hogy a feltüntetett MIB milyen szinten van megvalósítva a termékben. Nagyon sok esetben a gyártó „megfeledkezik" pl. a változó írhatóságáról (és így az eszköz vezérléséről SNMP felületen). Ugyancsak nehéz és bosszantó szituációkat idéz elő, ha a „támogatott" MIB-nek bizonyos változóit nem valósítják meg. Gyakori kerülőút a gyártó részéről, hogy ezekre a változókra mindig konstans 0 vagy NULL értékekkel válaszol, ami fejtörést okozhat (pl. egy speciális hiba számláló esetén). A korrekt gyártói viselkedés az lenne, ha ezekben az esetekben felhívnák a figyelmet a részleges implementációra, és ezeknél a változóknál hibaválaszt kapnánk.

3.5. Az SNMP protokoll

Az SNMP protokoll a fent bemutatott MIB változók elérését szolgálja. A protokoll a következő műveleteket tartalmazza **[RFC1157]** :

- Get, GetNext: a menedzsment állomás lekérdezi egy MIB változó értékét az SNMP ágenstől

- Set: a menedzsment állomás egy MIB változó értékét beállítja az ágensen.

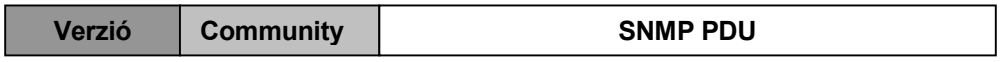
- Trap: a menedzselt objektumon futó SNMP ágens egy riasztást küld a központi állomás felé

A fenti műveletek alapján az az érzésünk támadhat, hogy az SNMP alapú menedzsment eszközei enyhén szólva korlátozottak. Meg kell azonban említeni, hogy a legtöbb MIB speciális változókon keresztül számos más beavatkozási lehetőséget biztosít. Ilyen lehet az eszköz újraindítása egy meghatározott változó 1-re állításával, vagy pl. Cisco router-ek esetén az

eszköz konfigurációs állományának letöltése egy TFTP szerverről szintén egy változó beállításával (a TFTP szerver IP címére). Sajnos ezek többnyire gyártófüggő megoldások.

Az SNMP protokoll szállítási rétege nem kötött, egyaránt használható megbízható vagy megbízhatatlan ill. kapcsolatorientált vagy kapcsolatmentes szállítási szolgáltatás igénybevételével. A gyakorlatban a legelterjedtebb szállítási rétege az UDP, melynek előnyei, hogy rövid és egyszerű üzenetváltás esetén kevés többlet erőforrást emészt fel (ellentétben a TCP-vel). Természetesen ebben az esetben csomagok elveszhetnek, többszöröződhetnek, de ezen a felhasználási területen ez nem okoz kijavíthatatlan hibákat. Az UDP esetén a protokoll két szolgáltatási portot használ. A GET/SET műveletek mindkét oldalon a 161-es portra érkeznek és portról indulnak. Az ágens a riasztásokat a menedzsment állomás 162-es portjára küldi.

Egy SNMP üzenet általános szerkezetét mutatja a 5. ábra. A verzió az általunk tárgyalt esetekben 1 értékű. Az ún. community a kliens azonosítására szolgál jelszó jelleggel.



5. ábra Egy SNMP üzenet általános szerkezete

Az SNMP PDU a korábban említett BER kódolással készül. Amennyiben az ágens a kérést nem tudja dekódolni, eldobja azt, nem válaszol. Ugyancsak válaszolatlanok maradnak azok a kérések, melyek a hozzáférés engedélyezése során akadnak fenn a rostán. Ha a kérés valamilyen más okból nem elégíthető ki, akkor hibaválaszt küld a hibahely megadásával. Egy válasz esetén az SNMP PDU a 6. ábran bemutatott módon néz ki.

6. ábra SNMP GetResponse PDU

A PDU típusa válasz esetén mindig GetResponse (2), a kérés azonosító pedig annak a tetszőleges értéknek a megismétlése, mely a választ generáló kérés jellegű PDU-ban szerepelt. Az azonosítóra a protokoll semmiféle megkötést nem tesz. A változók és étékeik lista (varbindlist) a kért változók azonosítóit (OID) és értékeik tartlmazza.

A SNMP kérés típusú PDU-k három kategóriába sorolhatók. Szerkezetük – melyet a 7. ábra mutat - azonban nagyon hasonlít.

7. ábra SNMP GetRequest, GetNextRequest, SetRequest PDU

Az egyszerű MIB változó lekérésnél a PDU típusa GetRequest (0). A változók listájában csak a kért változók azonosítói szerepelnek, a hozzájuk tartozó értékek NULL-ra vannak állítva. A kiszolgálás atomi természetű, vagyis ha bármelyik változóra vonatkozó kérést nem sikerül megválaszolni, akkor egy értéket sem fog visszaküldeni az ágens. A hibakód beállítása mellett a hibaindex a hibás változó sorszámára mutat a listában. Figyelnünk kell, hogy mind a kérés mind a válasz beférjen egyetlen UDP csomagba !

A GetNext (1) típusú lekérdezés azonos a hagyományos változó lekérdezéssel, de nem a kérésben megadott változó(k) értékeit, hanem rendre a közvetlenül rákövetkező változó(k) értékeit adja meg az ágens. A rákövetkezőt ilyenkor az számmal megadott objektum-azonosítók (OID) közötti lexikografikus rendezés adja meg. Ez gráfban egy olyan balról jobbra haladó mélységi bejárásnak felel meg, ahol a bejárás során csak a fa levelei érdekesek számunkra. A GetNext kérésekkel tehát egy ágens által megvalósított teljes

MIB bejárható, lekérdezhető a MIB előzetes ismerete nélkül. Ezt a bejárást hívják SNMP walk-nak.

Az SNMP értékek beállítása során egy SetRequest (2) csomagot küldünk, mely az egyszerű lekérdezéssel azonosan működik, de a változó lista érték elemeit beállítjuk.

A ágens által generált riasztások Trap PDU(4)–ként jutnak a munkaállomáshoz. A PDU-ban szereplő mezők csak korlátozott hibadetektálást tesznek lehetővé. A gyakorlatban a trap fogadása után a menedzsment állomás további SNMP kérésekkel határolja be a hiba okát. A trap szerkezetét mutatja a 8. ábra.

8. ábra SNMP Trap PDU

Az trap speciális mezői **[RFC1215]** közül érdekes a gyártóra vonatkozó azonosító (ennek a speciális trap típusnál van jelentősége), az általános mező (mely előre definiált gyakori hibakódokat tartalmazhat. Ha az általános mező értéke enterpriseSpecific(6), akkor egy gyártófüggő hiba kódot tartalmaz a speciális mező. Az SNMP trap-ek gyártófüggő változatait is ASN.1 nyelven adják meg.

Az ágens által kezelt MIB változók elérése során el kell dönteni, hogy a hozzáférés engedélyezett-e a hivatkozott objektumhoz (változóhoz). Erre az SNMPv1-ben elég korlátozott módszerek vannak, az SNMPv2 ill. a protokoll 3. változata ezen a téren jelentősen előrelépett. Most csak az eredeti biztonsági modellel foglalkozunk. A hozzáférés engedélyezését a következő információk döntik el:

- Manager állomásra vonatkozó információk: munkaállomás IP címe és a kérésben szereplő SNMP „community string" – jelszószerű token, melyet a protokoll kódolatlan formában (cleartext) visz át a hálózaton

- Ágens konfigurációs beállításai. Ezek határozzák meg, hogy az adott kliens adott community string felmutatásával mit és milyen műveletekkel ér el.

- A hivatkozott MIB változóra vonatkozó ACCESS attribútum

Az ágens konfigurálása során a kliensre vonatkozó adatok függvényében meghatározhatjuk, hogy a MIB fának melyik részfáját érheti el a kliens. Ezt nevezzük MIB nézetnek (MIB View). Ugyanígy meghatározhatjuk, hogy ezen a részfán milyen műveletek végezhetők el (READ-ONLY, READ-WRITE). A hozzáférést a részfában szereplő objektum egyedi attribútuma megváltoztathatja (szűkítheti). A nézetet és a hozzáférési módot SNMP profilnak (SNMP community profile) nevezik.

3.6. Az SNMP korlátai

Mivel várhatóan középtávon az SNMP marad az igazán elfogadott hálózat menedzsment protokoll, hasznos összefoglalnunk és szem előtt tartanunk komolyabb hiányosságait.

- A korábban számszerű példával is érzékeltett skálázhatósági korlátok lehetetlenné teszik igazán nagy hálózatok központi menedzsmentjét. A skálázhatósági probléma egyik fő oka, hogy minden egyes SNMP változó értékének lekérdezéséhez a kéréseket minden egyes változóra meg kell tennünk.

- Az SNMP tarp-ek nyugtázatlanok (és az UDP szállítási réteg sem segít ezen). Így kritikusabb területeken nem lehet ezekre bízni a felügyeletet.

- Az SNMPv1 biztonsági lehetőségei korlátozottak, és könnyen megkerülhetők (pl. hálózat lehallgatásával)

- Az SNMP műveletek köre rendkívül szűk. Speciális parancsok végrehajtása meghatározott változók írásával történhet, azonban ezek a mechanizmusok nem szabványosítottak

- A MIB modell, és a tisztán skalár alapú információ kezelés nem minden esetben kielégítő, sokszor nagyon nehézkessé teszi komplexebb feladatok elvégzését.

- Az SNMP protokoll nem támogatja a menedzsment állomások közötti kommunikációt és információcserét. Ezzel a lehetőséggel a skálázhatósági problémák nagy része orvosolhatóvá válna.

3.7. A CMIP protokoll

Az eredeti elképzelések szerint a CMIP lett volna az a protokoll, mely felváltja az SNMP-t a 90-s évek elején. Sokan a kezdeti implementációk problémáinak tudják be, hogy mind a mai napig a CMIP nem tudott jelentősebb mértékben elterjedni. Bár több kereskedelmi operációs rendszerhez (pl. Sun Solaris) kaphatók CMIP könyvtárak, fejlesztői eszközök, ezek elsődleges alkalmazási területe ma is a kutatás és hosszabb távú fejlesztések.

Az SNMP eredeti 5 protokoll üzenetével (PDU) ellentétben a CMIP 11 primitívet (Event, Create, Delete, Action) támogat, melynek egyik fő oka, hogy a menedzselt objektumok szerkezete sokkal összetettebb az SNMP skalár megközelítésénél. Ezek az objektumok rendelkeznek attribútumokkal, műveletekkel (metódusokkal), és eseményekkel (melyeket az objektum generálhat).

A biztonság terén is jelentős újításokat hozott, melyeket finomabb hozzáférés szabályozási, hitelesítési és naplózási lehetőségeket biztosítanak.

A CMIP inkább aggregált információk lehívására orientált, és a riasztásoknak sokkal nagyobb szerepük van, ehhez természetesen a CMIP-et megvalósító ágenseket kell nagyobb intelligenciával felruházni.

A CMIP architektúra részletesebb bemutatása megtalálható Stallings **[Sta_99]** könyvében.

4. JELENLEGI MENEDZSMENT RENDSZEREK ÁTTEKINTÉSE

Az előző fejezetben ismertetett menedzsment protokollok önmagukban még nem elegendőek a hálózat felügyeleti feladatok ellátásához. Ehhez olyan keretrendszereket kell még készíteni, mely a menedzsment protokollok ismeretén túl képes a menedzsment információ tárolására, rendszerezésére, értelmezésére, és mindezt a felhasználó (üzemeltető) számára érthető és szemléletes módon megjeleníti **[Tre_95]** .

Ezeket a keretrendszereket a rendszermenedzsment világában platformoknak hívják. A ma igazán elterjedt platformokkal (HP Openview, Tivoli, BMC, stb.) szemben támasztott követelmények: a széleskörű alapszolgáltatások, és a nyílt architektúra, független gyártók moduljainak integrálhatósága. Ezek a keretrendszerek szinte minden pontjukon elérhetőek a hozzájuk adott programozói felületek (API.-k) segítségével, bár az integrálhatóság nem kizárólag az ezek segítségével történik.

A továbbiakban röviden áttekintjük, hogy melyek egy ilyen menedzsment platform alapszolgáltatásai, majd egy konkrét – és hazánkban is igen elterjedt – eszközzel, a HP OpenView rendszerrel fogunk foglalkozni. A célom, hogy bemutassam ezen rendszerek jellegzetes vonásait, majd néhány problémára, hiányosságra felhívjam a figyelmet.

4.1. Menedzsment platformok szolgáltatásai

Minden komolyabb menedzsment platformmal szemben támasztott alapkövetelmények:

- Szabványos menedzsment protokollok ismerete. Ezek elsősorban egy programkönyvtárban vannak, melyeket egyrészt a keretrendszer, másrészt a külső modulok számára biztosított API-n keresztül a külső modulok vesznek igénybe. Alapkövetelmény az SNMP (különböző verziói) támogatása, de nem ritkaság a CMIP implemetálása sem. Érdemes megjegyezni, hogy vannak nem kifejezetten menedzsment célra kifejlesztett protokollok is, melyeket mégis fel lehet (kell) használni a felügyelet során, így ezek támogatottsága is egyre nő ezekben a rendszerekben. A legfontosabb ilyen technológiák: *SunRPC* (távoli eljáráshívás), *CORBA* (távoli objektumok elérése – elosztott objektum orientált rendszerek kialakítása programozási nyelvtől függetlenül), *RSH/TELNET* (bejelentkezés – távoli eszköz parancssori elérése, melyet általában automatizált módon kezel a menedzsment álllomás, pl. Expect programcsomag segítségével), *OSF/DCE* (Az RPC-vel rokon távoli eljáráshívási keretrendszer)

- MIB importálási lehetőségek. Gyártótól származó MIB leírások értelmezése és használata. Itt csak azt várjuk el a rendszertől, hogy szintaktikailag értelmezze ezeket a külső ASN.1 leírásokat, az egyes változók mélyebb értelmezése külső modulok fejlesztésével valósul meg.

- Szabványos MIB-ek értelmezése, felhasználása. Elvárható, hogy a menedzsment rendszer a legfontosabb MIB változókat értelmezni is tudja, s így kezelni tudjon olyan információkat, mint pl. adott eszköz interfészeinek száma, forgalma, állapota, stb.

- Hálózati feltárás. Nagyobb hálózatok esetén nagyon fontos, hogy a keretrendszer képes legyen a hálózat automatikus feltérképezésére, mely egyrészt a hálózati eszközök felismeréséből, másrészt a köztük levő topológiai viszonyok felderítéséből áll. Mivel ezt a feladatot az IP hálózatok csak korlátozott mértékben segítik, itt nagyon sok múlik a menedzsment szoftveren és a felderítés (discovery) során alkalmazott heurisztikákon. Ezzel a témával később egy külön fejezetben fogunk még foglalkozni.

- Grafikus megjelenítés, térképek. Az előző pontban felismert hálózatot átlátható módon kell megjeleníteni. Nagyobb hálózatok esetén elengedhetetlen a hierarchikus szervezés, ahol a különböző felbontású térképek egymás alatt több síkban helyezkednek el. A legtöbb platform lelke ez a hálózati térkép, melyről a legtöbb művelet kezdeményezhető, ill ahol a szokatlan események megjeleníthetők. Ugyancsak hasznos funkció, ha a hálózati objektumok a topológiai viszonyokon túl logikai csoportokba is szervezhetők, mellyel funkció szerinti nézeteket készíthetünk (pl. Tivoli Netview esetén az ún. SmartSet-ek)

- Riasztási képességek. Egyrészt képesnek kell lennie a rendszernek arra, hogy külső SNMP üzeneteket (trap-eket) fogadjon és értelmezzen, másrészt az menedzsment rendszertől is elvárjuk, hogy bizonyos kritikus – vagy általunk definiált (pl. egy változó értékére megadott küszöb átlépése) – esetekben maga generáljon ilyen riasztásokat. További elvárás, hogy a riasztásokat naplózza, és a riasztásokat szűrje (eseménykorreláció). Kritikus esetekben elvárjuk, hogy a riasztást eszkalálja a megfelelő személyek felé (pl. ticket küldése e-mail segítségével, SMS küldése, ablak feldobása a felhasználói felületen, hangjelzés, stb.)

- Adatgyűjtés, adatbázisok. A rendszertől elvárjuk, hogy a megfigyelt hálózat elemeiről folyamatosan adatot gyűjtsön, és ezeket a statisztikai információkat saját vagy külső adatbázisban tárolja. Ennek alapján a rendszer válaszolni képes olyan alapvető kérdésekre, mint egy adott gép százalékos elérhetősége az elmúlt hónapban, hálózati szegmensek forgalmának alakulása (grafikonon). A legtöbb esetben a relációs adatbázis formát használják erre a célra, és támogatják az elterjedtebb RDBMS-ekkel való integrációt (pl. Informix, Oracle, stb.)

- Nyílt architektúra. A keretrendszer kibővítésével a következő szakaszban foglalkozunk.

- Elosztott menedzsment támogatása. Itt a korábban ismertetett skálázhatósági problémákat kell megoldani. Ezen a következő szolgáltatásokkal lehet segíteni: tartomány menedzsment (a teljes hálózat menedzsment szempontok szerinti partíciónálása), menedzserprogramok között kommunikáció, hierarchikus menedzsment támogatása (az események, információk lokális szűrésével).

4.2. Nyílt architektúrával szembeni elvárások

A fenti követelmények teljesülése még mindig nem elégséges egy valódi hálózat hatékony menedzseléséhez. Ehhez még olyan egyedi modulokkal kell kiegészítetnünk a keretrendszert, mely illeszkedik az alkalmazott eszközök – gyártófüggő – sajátosságaihoz. Az architektúra több ponton is nyílt kell, hogy legyen. Most a legfontosabb hozzáférési pontokat tekintjük át.

A külső MIB definíciók értelmezése, beolvasása az első lépés az egyedi képességek támogatása felé. Ezek hatására a speciális változók típusa és neve ismert lesz az alkalmazás számára. További fejlesztések nélkül is előírhatjuk már ezen változók adatgyűjtését, ill. trap-ek fogadását.

A kiterjeszthetőség legfontosabb eszköze a keretrendszer által szolgáltatott API gyűjtemény. Ezek legtöbbször 'C' nyelvű programkönyvtárak. Ahhoz, hogy valóban integrálhassuk saját moduljainkat a keretrendszerrel, a következő felületeket kell biztosítani: grafikus elemek elérése (térkép szimbólumok, dialógus ablakok, stb.), objektum és topológiai adatbázis, menedzsment protokollok és MIB adatbázis, statisztikai adatbázis és funkciók. Ha ezek a rendelkezésünkre állnak, akkor valóban az adott feladatra összpontosíthatunk: az egyes menedzsment információk magasabb szintű értelmezésére.

Sokszor az egyedi változók értelmezését csak olyan szinten szeretnénk bővíteni, hogy bizonyos logikailag összetartozó információk táblázatos formában, vagy grafikonon legyenek megjeleníthetők. Ehhez túl komplex feladat C nyelvű modulokat fejleszteni, ezért egyes keretrendszerek lehetőséget biztosítanak ilyen jellegű eszközök fejlesztésére (pl. MIB Tool Builder a Tivoli Netview esetén). A grafikus felületen megtervezett modult a rendszer általában valamilyen szkript nyelven tárolja. Menedzsment környezetben igen gyakori a Tcl nyelv alkalmazása.

Az általunk, vagy független gyártó által készített modulokat valahogyan elérhetővé kell tenni a felhasználói felületről. Kézenfekvő megoldás az egyes menük kiterjesztése, ill. bizonyos felhasználói eseményekhez kapcsolt funkciók módosítása. Ehhez szintén felületet kell biztosítania a keretrendszernek. Sok esetben (pl. HP OpenView) ez ún. regisztrációs file-ok segítségével valósul meg, melyekben leírjuk, hogy a modulunk melyik menübe kerüljön, milyen nevet kapjon, ill. milyen tulajdonságokkal rendelkező objektumok esetén legyen elérhető. A regisztrációs állományokat egy speciális könyvtárba másoljuk, és a rendszer elindításakor ezek alapján fogja felépíteni a menürendszerét.

4.3. A HP OpenView NNM platform

A HP OpenView platform ma már a menedzsment számos területén megoldást kínál: a desktop menedzsmenttől a betörések észleléséig (Intruder Detection System – IDS) számos termékkel rendelkezi. Minket most a hálózatmenedzsmentet megvalósító Network Node Manager érdekel. Elterjedtsége mellett azért esett erre a választásom, mert a mai robosztusabb menedzsment platformok szinte minden jellegzetes vonásával rendelkezik. Eredetileg UNIX környezetbe szánták (HP-UX, Solaris, stb.), ma már létezik azonban Windows NT-n futó változata, bár a rendszer megőrizte UNIX-os filozófiáját. Az ismertetés során csak a rendszer komponensei (processzek, adatbázisok) és a kapcsolataik ismertetésére szorítkozom.

A NNM működése **[HP_00]** két rétegben valósul meg. Az első szinthez tartozó folyamatok folyamatosan futnak, a hálózat felügyelete (események fogadása, adatgyűjtés) tulajdonképpen ennek a rétegnek a feladata. A második szint a felhasználói felület ill. az onnan elérhető funkciókért felelős. Mindkét rétegben megkülönböztetünk ún. előtérben futó (foreground) és háttér (background) folyamatokat. A felhasználó csak az előtér folyamatokat látja ill. indítja el.

Az első réteg szerkezetét mutatja a 9. ábra. Az ábrán szögletes dobozok mutatják a háttérfolyamatokat, az előtérben futó programok ellipszis alakú keretben vannak. A konfigurációs file-ok két vonal között szerepelnek, az adatbázisok pedig a szokásos korong jelölést kapták. A nyilak az információáramlás irányát mutatják (a kommunikáció fizikai megvalósítása különböző lehet).

A felhasználó öt előtérprogrammal irányítja a teljes rendszert. (ovstart – elindítás, ovstop – megállítás, ovstat – rendszer folyamatainak ellenőrzése, ovpause, ovresume). A réteg összehangolt működéséért az ovsmpd folyamat felelős. További háttérfolyamatok:

- netmon: a hálózat folyamatos monitorozását végzi új eszközök, hálózatok, topológiaváltozások után kutatva

- httpd: az újabb OpenView rendszerek esetén a Web felület biztosításáért felelős

- ovactiond: kritikus események, riasztásokhoz kötött műveletek elvégzése (pl. SMS küldés), információkat a pmd-től kap

- ovalarmsrv: az események listáját, attribútumait szolgáltatja a Java alapú esemény-böngészőnek

- ovcapsd: hálózati eszközök, állomások DMI (Desktop Management Interfész) és Web alapú menedzselhetőségét ellnenőrzi, keresi

- ovrepld: elosztott működési módban az objektum adatbázis szinkronizációjáért felelős.

- ovrequestd: időzített, rendszeres beszámolók készítése az összegyűjtött információk alapján (divatos szóval élve: data warehouse funkciók)

- ovtopmd: a netmon folyamat segítségével a topológiai adatbázis karbantartása, frissítése

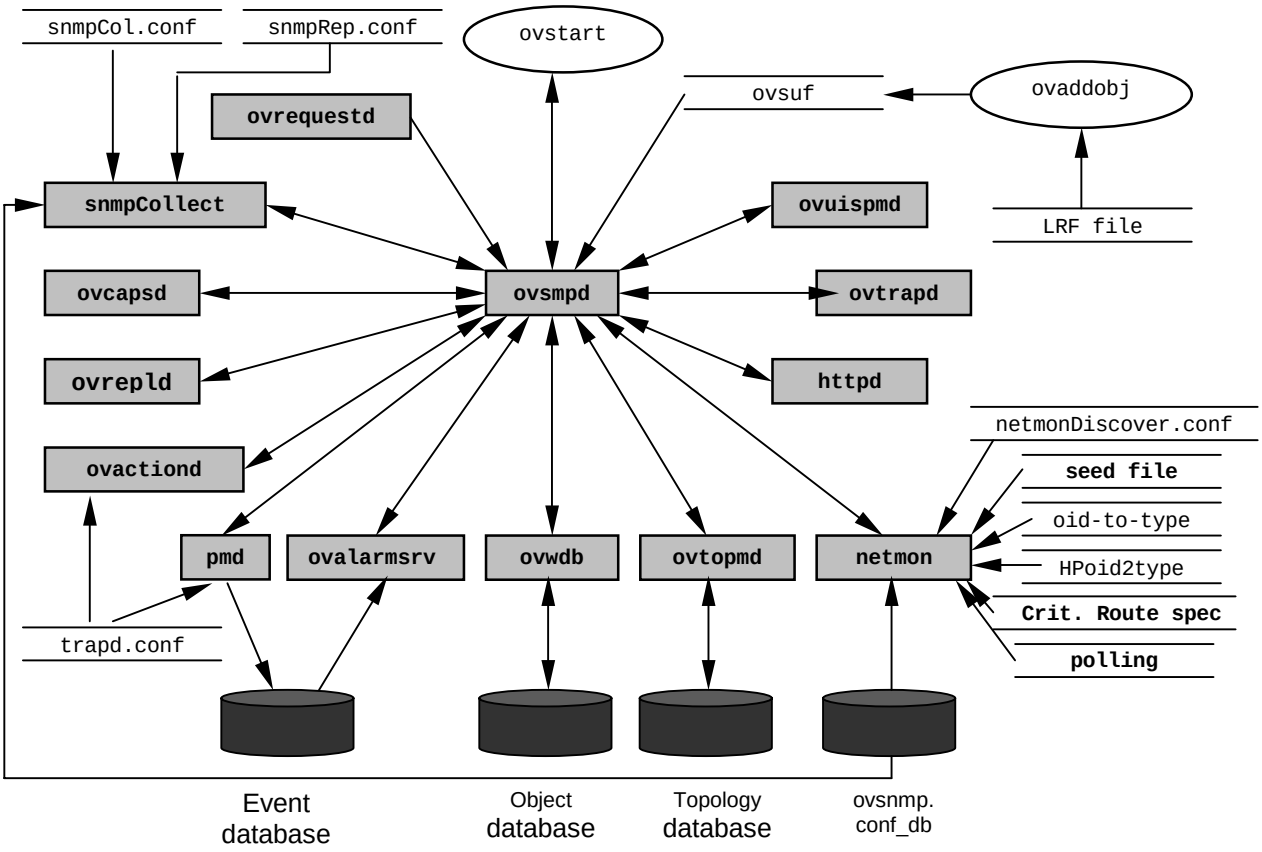
- ovtrapd: SNMP trap üzernetek fogadása és továbbítása a pmd folyamat felé

- ovuispmd: a második réteg folyamatainak vezérlését biztosítja

- ovwdb: az objektum adatbázis karbantartásáért felelős

- pmd: riasztások fogadását, naplózását, esetleges továbbítását (más alkalmazások felé) végzi

- snmpCollect: folyamatos adatgyűjtést végez, statisztikai adatbázist épít, küszöbértékek átlépése esetén riasztást küld a pmd folyamat felé



9. ábra HP OpenView első réteg, folyamatos működés

Az előtérfolyamatok az ovw parancs hatására indulnak el. A legfontosabb ezek közül maga az ovw folyamat, mely a grafikus felhasználói felületért felelős. Más folyamatok egyes menüpontok kiválasztására indulnak. Ezek közé tartoznak az általunk megírt modulok is.

Érdemes még megemlíteni egy harmadik réteget is, mely az újabb NNM verziókban jelent meg, s mely Web felületet biztosít a menedzsment feladatok elvégzéséhez. A httpd folyamat különböző CGI jellegű programokkal szolgálja ki a böngésző felől érkező kéréseket, mely programok a fent bemutatott háttér folyamatoktól és adatbázisokból nyerik a szükséges információkat.

4.4. Problémák

A klasszikus menedzsment platformok néhány olyan problémával rendelkeznek, melyekre a dolgozat hátralevő részében megoldásokat keresünk. A legtöbb problémát a gyártók is érzik, így a bemutatásra kerülő technológiák jó részét időközben megvalósították ezekben a termékekben.

A „jó" keretrendszerrel szemben támasztott követelmények között volt a skálázhatóság biztosítása. Ezt azonban a legtöbb termék ma még elég körülményesen vagy egyáltalán nem támogatja. A megvalósítások többnyire teljesen egyedi technológiákat alkalmaznak, elég elképzelhetetlen ma még egy olyan menedzsment környezet, ahol az egyes alhálózatokat különböző menedzsment platformok kezelik, és ezek képesek az együttműködésre. A skálázhatósági problémákat pedig jól ismerjük. Elvétve találkozunk azonban már itt is ígéretes megoldásokkal (pl. Cisco Wan Manager), ahol a teljes menedzsment rendszer egy proxy ágensként (is) viselkedik.

További probléma, hogy a központi menedzsment állomás az üzemeltető személyzetet sokszor helyhez köti. Elképzelhető, hogy helyszíni hibaelhárítás közben is szükség lenne a menedzsment rendszer használatára, ez azonban nehezen megoldható (X Window alapú programok esetén körülményes és nem biztonságos, MS Windows esetén még komplikáltabb). Az informatika más területein is hódít a Web alapú felhasználói felület alkalmazása, mely itt is gyógyír lehetne a problémára.

A hálózatmenedzsment egyik legdinamikusabban fejlődő területe az események közötti összefüggések feltárásával foglalkozik. Az eseménykorreláció elengedhetetlenül fontos sok csomópontot tartalmazó hálózatok esetén. A feladat minél tökéletesebb megoldása itt is heurisztikus megközelítést igényel, ezért ezzel egy külön fejezetben foglalkozunk.

Még a nyílt menedzsment platformok is rendelkeznek olyan elemekkel, melyek teljesen egyediek, mégis szükség lenne a külső (szabványos protokollon keresztüli) hozzáférésre. Egyik jó példája ennek a hálózati erőforrások (objetumok) és topológia adatbázisa. A meglévő eszközpark adatait elképzelhető, hogy más alkalmazásokban is szeretnénk használni (pl. számlázás, biztonsági funkciók, konfigurációk nyilvántartása), így áldásos lenne ezeket az információkat nem saját (egyedi szervezésű) adatbázisokban tárolni. A címtár rendszerek kifejezetten jó megoldásnak tűnnek ezen a téren Ezen belül is az LDAP protokoll számíthat széles körű alkalmazásra. A menedzsment rendszerek ezen a téren még gyerekcipőben járnak, pedig a címtárak más funkciókat is segíthetnének (pl. korrelációvizsgálat – lásd. később, elosztott menedzsment).

5. ÚJ LEHETŐSÉGEK

A következő fejezetekben olyan eszközöket ill. technológiákat szeretnék bemutatni, melyek egy részét napjainkban kezdik szélesebb körben alkalmazni a menedzsment feladatok megoldása során, vagy előnyös tulajdonságuk miatt alkalmasnak találom ilyen jellegű feladatok megoldásához. Az egyes technológiák bemutatása mellett nagy hangsúlyt fektetek arra, hogy megvilágítsam: miért és milyen módon lehet ezeket hatékonyan bevonni a felügyeleti munkába.

Néhány elképzelés teljesen újszerűnek tűnt, amikor először kezdtünk foglalkozni alkalmazhatóságával a hálózat felügyelet terén, ilyen volt többek között a címtárak alkalmazása az eszközök konfigurációs és topológiai információinak tárolására. Az elmúlt lassan egy év során ezek a megoldások szinte kivétel nélkül megjelentek a kereskedelmi hálózatmenedzsment termékekben. Így bár elképzeléseink jelenleg már nem tűnnek úttörő jellegűeknek, mégis örömmel látjuk, hogy életképeseknek bizonyulnak.

A most ismertetésre kerülő három terület a WEB alapú menedzsment, az LDAP címtárak használata ill. a riasztások közötti korrelációk vizsgálata.

A következő három rész egymástól függetlenül olvasható, ezekben az egyes területek alaposabb leírását találhatjuk.

6. WEB ALAPÚ HÁLÓZATMENEDZSMENT

A hálózatmenedzsment területén felbukkanó új paradigmák egyike a Web alapú hálózatmenedzsment *(Web Based Management Applications – WBMA)*. A fogalom elsősorban azt a törekvést fedi, mely a meglévő robosztus és sokszor platformfüggő hálózatmenedzsment alkalmazásokat egy sokkal nyíltabb és szinte bárhonnan elérhető, irányítható architektúrával kívánja kiváltani. A számítástechnika/informatika számos területén sikerült már bizonyos alkalmazásokat HTTP-HTML alapokra átültetni, mint például az adatbáziskezelés, címtárszolgáltatás.

6.1. A WEB alapú megközelítés jellemzői

A Web alapú architektúrák előnyei:

- Kliens/szerver rendszerek: ezen belül az ún. *flat server – thin client* elképzelést valósítják meg, ami annyit tesz, hogy az alkalmazás érdemi része (pl. az adatbázis és az azt kezelő programkód) a kiszolgálón fut. Ennek előnyei a könnyű karbantarthatóság és az alkalmazáshoz való hozzáférés egyszerűsítése. (A klasszikus hálózatmenedzsment platformok esetén sokszor fizikailag a menedzsment alkalmazást futtató munkaállomás mellett kell ülnie az azt használó operátornak)

- A WWW technológiáiból származó előny, hogy az ügyfél oldalán nem szükséges különleges ill. egyedi program telepítése, kizárólag egy böngészőre van szükség az alkalmazás eléréséhez. Ezen technológiák teljes mértékben nyitottak, szabványosításukkal nonprofit és gyártófüggetlen szervezetek foglalkoznak *(pl. IETF, W3C)*.

A Web alapú megoldások rendelkeznek néhány hátrányos tulajdonsággal is:

- Az alkalmazás működése nehézkes, szakaszos. Ennek oka, hogy a kliens (böngésző) és az alkalmazás közötti kapcsolat nem állandó. Mindenki bosszankodott már, hogy egy hosszú űrlap kitöltése után kaptunk hibaüzenetet helytelenül kitöltött mezőkről. Ennek az oka, hogy a felhasználói inputot értelmező (kiszolgáló oldalon futó) program csak a teljes űrlap kitöltése után kezd el futni, az ún. *submit* művelet hatására. Érdekes felfedezni a működésben rejlő hasonlóságot az IBM 3270 terminál és az IBM mainframe-eken futó alkalmazásokkal.

- A Web alapú alkalmazásfejlesztők számára a legnagyobb fejtörést az alkalmazás állapotmentessége okozza. Ez a kapcsolatmentes működésből adódik. Minden egyes submit művelet hatására a kliens és a kiszolgáló új kapcsolatot nyit, és a kiszolgálón futó alkalmazás is sok esetben újra indul. Erre a problémára több megoldás is született, ilyenek az ún. *cookie*-k, vagy a Web kiszolgálóba integrálható programmodulok *(ISAPI, NSAPI, Apache modul)*

A fentieket figyelembe véve kell mérlegelni, hogy egy adott feladat alkalmas-e illetve igényli-e a Web alapú megvalósítást.

6.2. Két alapvető megközelítés

A hálózatmenedzsment területén a Web alapú megközelítésnek két lehetséges útja van. Mindkettőre léteznek már működő (ipari, kereskedelmi) alkalmazások. A két lehetséges megközelítés:

1. Egy központi alkalmazással valósítják meg az árjárást a böngészők és menedzselt objektumok között. Olyan alkalmazásról van tehát szó, mely egyrészt egy WWW kiszolgáló funkciót lát el, másrészt képes a hálózatmenedzsmenthez szükséges protokollok használatára *(pl.*

SNMP, RPC). A megoldás előnye, hogy képes összefüggően kezelni a hálózatot, rendszeres statisztikai adatbázis építhető a hálózati hibákról és forgalomról, és jól skálázható. További előnye, hogy az eszközök oldalán nem igényel módosítást, újabb programok telepítését, hanem kihasználja a már meglévő és szabványosnak tekinthető menedzsment protokollokat.

2. Sok gyártó választja azt az utat, hogy eszközébe olyan (szoftver) modult épít, mely közvetlenül képes a HTTP, HTML kezelésére, így a böngésző programmal ezek az eszközök közvetlen elérhetők, konfigurálhatók. A megoldás lehetővé teszi egyedi eszközök egyszerű kezelését, de egyáltalán nem skálázható.

Amikor arról beszélünk, hogy a meglévő robosztus menedzsment platformokat ültetjük Web köntösbe, akkor világosan látszik, hogy az első utat kell választanunk. A második megközelítéssel nagyon kényelmesen lehet egyedi eszközöket kezelni, több eszköz esetén azonban problémaként jelentkezik, hogy a HTTP/HTML alapú menedzsment nem szabványosított ill. használata nem automatizálható. Mivel a különböző gyártók eszközeinek Web-es felülete jelentősen eltérő, ezen interfészeket igen nehézkes lenne központi menedzsment alá vonni.

6.3. Biztonsági kérdések

A Web felület – ha nem is annyira kényelmes, mint az operációs rendszerfüggő alkalmazások – nagy előnye tehát a távoli, tetszőleges helyről lehetővé váló hozzáférés. Ez azonban sok biztonsággal kapcsolatos kérdést vet fel. Míg a klasszikus menedzsment állomást vasrács mögé tehettük, addig a Web felület esetén egyedül az alkalmazásnak kell gondoskodnia a hozzáférés szabályozásáról. Ugyancsak kritikus kérdés a menedzsment információk (pl. router jelszavak) áthaladása a hálózaton. A menedzsment rendszer illetéktelen kézbe kerülése súlyos helyzetekhez vezethet.

Sajnos a HTTP protokoll nem ad a fenti problémákra megnyugtató választ. A fogalom titkosítatlan, a felhasználó azonosításához szükséges jelszavak úgyszintén. A megoldást a Netscape által kifejlesztett és azóta az IETF által szabványosított SSL (Secure Socket Layer) nyújtja.

Az SSL nem a HTTP kiterjesztése, nem is kizárólag Web-specifikus, más TCP alapú protokollokhoz is használható. Az SSL a HTTP-nél alacsonyabb szintű protokoll, biztonságos csatornát képes létrehozni két végrendszer között: end-to-end titkosítást, digitális aláírást, kliens és szerver azonosítást egyaránt támogat. A kapcsolatba lépő felek igazolhatják az azonosságukat a számukra kiállított digitális igazolásokkal.

A kapcsolat kezdetén – azonosítás után - a kliens és a szerver meghatároz egy titkos kulcsot, majd ezzel a kulccsal szimmetrikus rejtjelezéssel cserélnek információt.

Az SSL-hez használt digitális aláírások az X.509 szabvány alapján épülnek fel. Ebben a modellben a rendszerben (nem feltétlenül a hálózaton) egy kitüntetett bizalmi központ létezik (Certificate Authority), melyben –és csak benne -mindenki megbízik, vagyis az általa kiállított azonosítókat fenntartás nélkül érvényesnek tekinti.

Az SSL véd a lehallgatástól, a beékelődéstől (Men-In-the-Middle), illetéktelen hozzáféréstől, így ideális megoldás lehet a Web alapú menedzsment területén. Elterjedését hátráltatja a szükséges háttér infrastruktúra (Public Key Infrastrucure – PKI), valamint az Egyesült Államok (korábbi) kriptográfiai exportkorátozásai.

6.4. Kapcsolódó technológiák

A Web felület kialakításához számos eszköz, technológia áll a rendelkezésünkre. Rövid áttekintésük során megmutatom, hogy melyek azok a módszerek, melyek a menedzsment terén is segítésünkre lehetnek, ill. ezeket a menedzsment rendszer mely területén lehet hatékonyan alkalmazni.

A statikus file-ok után az első komolyabb – és már az alkalmazások irányába mutató – megoldás a CGI felület megalkotása volt, ahol a dinamikus (felhasználói interakció alapján kialakított) tartalmat a WWW szerver által indított programok szolgáltatták. Erőforrásigénye mellett komoly fejtörést okoz a fejlesztőknek az állapot mentése ill. beállítása két HTTP kérés között. (Az ún. cookie technológia elsősorban ezt kívánja segíteni). Előnye a programozási nyelvtől független felület.

A CGI-hez hasonló megoldás a HTML oldalakba ágyazott kódrészletek WWW szerver általi végrehajtása. Itt tehát nem a program tartalmaz HTML elemeket, hanem a HTML file tartalmaz kódrészleteket. A CGI-nél említett problémák részben itt is jelentkeznek. Természetesen a kódrészletek végrehajtásához a szerver programnak ismernie kell a kódhoz használt programozási nyelvet. A legelterjedtebb ilyen technológiák: PHP, SSI (Server Side Include).

A Web technológiák kapcsán előszeretettel emlegetett Java nyelv több lehetőséget kínál. A szerver oldalon ún. servlet-ek segítségével általunk írt kódot ültethetünk a WWW szerverbe, így az állapotmegőrzés egyszerűsödik. A böngésző oldalán az ún. applet-ek komolyabb feladatok kliens oldali végrehajtását teszik lehetővé, ezzel sokkal természetesebbé téve az alkalmazás felhasználói felületét (nincs 'submit' jellegű szaggatott interakció). Az applet-ek ezért összetetebb menedzsment rendszerekben minden bizonnyal megjelennek. Érdemes még megemlíteni a szerver oldalon az ún. JSP (Java Server Pages) technológiát, mely a java kódot HTML oldalakba ágyazza. A

Java platformfüggetlen jellege - ill. az a cél, hogy beágyazott rendszerek programozási eszköze legyen – széleskörű alkalmazási lehetőségekkel kecsegtet a menedzselt eszközök oldalán. Hosszabb távon elképzelhető, hogy az elosztott menedzsmentet ill. magukat a menedzsment protokollokat részben kiváltják a Java elosztott funkciói (objektum perzisztencia, távoli objektumelérés)

A Java mellett egy másik objektumorientált és elosztott architektúra a programozási nyelvtől független CORBA, melynek Web-es kapcsolatát a WAI (Web Application Interface) szabvány rögzíti. Bár több WWW szerver (pl. Netscape Enterprise Server) támogatja ezt a felületet, e terület messze nem olyan dinamikus, mint a Java alapú megoldásoké.

A mai beágyazott rendszerekben – és a kísérleti megvalósításban is – használt megoldás szerint a dinamikus tartalmat előállító kódot közvetlenül a WWW szerverbe integráljuk, illeszkedve a szerver programozási nyelvéhez, architektúrájához. A legelterjedtebb WWW szerver programozói felületek: a Netscape szervereknél használt NSAPI (Netscape Server Application Programming Interface) ill. a Microsoft termékek ISAPI (Information Server Application Programming Interface) felülete. Az általam kialakított WWW felület a Tcl Web Server saját Tcl nyelvű felületét használja.

7. CÍMTÁRKEZELÉS: LDAP

7.1. A címtárszolgáltatásokról

A címtár rendszerek olyan hálózati szolgáltatások, melyekben különböző objektumokra (személyek, eszközök, stb.) vonatkozó információkat lehet hatékonyan tárolni, ill. visszakeresni. Hasonlítanak a valós élet telefonkönyveihez vagy leltárakhoz.

A címtárszolgáltatások már elég régóta jelen vannak a számítástechnikában. Ilyen meglévő technológiák: *NDS* (Novell), *DNS* (Internet), *NIS* (Sun). Sajnos, ezek egyike sem tudta elfogadtatni magát teljes körűen az informatikai világgal, aminek fő oka erősen gyártófüggő voltuk vagy speciális felhasználási területük (pl. DNS).

Az *OSI* protokollcsalád szintén szabványosított egy címtár rendszert. A szabvány az *X.500* jelölést kapta. Ennek egyetlen szépséghibája – más OSI szabvány társaihoz hasonlóan – a meglehetősen bonyolult és nehezen implementálható architektúra. Az X.500-ban rejlő lehetőségeket felismerve a michigan-i egyetemen kidolgozták ennek egy egyszerűsített változatát, melynek az *LDAP (Lightweight Directory Access Protocol)* nevet adták **[RFC1777]** . Elkészítettek egy átjárót is, mely segítségével X.500 adatbázisokat lehetett elérni LDAP protokoll segítségével.

Az LDAP annyira használhatónak bizonyult, hogy hamarosan az *IETF* is belekezdett szabványosításába, és az átjáróból önálló kiszolgáló lett saját adatbázissal. Ma a legtöbb szoftvergyártó letette már voksát az LDAP mellett,

így várható, hogy ezen a téren az LDAP valóban közös nyelvvé válik hamarosan.

7.2. Az LDAP alapjai

Mik az LDAP legfontosabb jellemzői? Elsősorban úgy kell tekintenünk egy LDAP szerverre, mint egy olyan speciális adatbázisra, melyet az esetek döntő többségében csak lekérdezésre használnak, és a tárolandó objektumok mérete viszonylag kicsi. Ebből következően az írás műveletek meglehetősen lassúak, és általában nincs biztosítva a konkurens hozzáférés szabályozása sem. Egyáltalán nem rendelkezik olyan tranzakció-kezelő funkciókkal, mint az SQL adatbázisok. A keresés, lekérdezés jellegű műveletek viszont rendkívül gyorsak. A legtöbb LDAP szerver biztosít valamilyen replikációs mechanizmust, vagyis a címtár redundáns tárolását ill. terhelésmegosztást több szerver között. Az ilyen esetekben egy ún. master és több slave (vagy backup) kiszolgáló kerül telepítésre. Az LDAP támogatja az ún. hivatkozás-átirányítást (referals), melyek segítségével több LDAP adatbázis összekapcsolható egy nagyobb rendszerré. A fenti tulajdonságok alapján kell dönteni, hogy mely feladatokkal célszerű LDAP-val és mely funkciókat kell SQL adatbázisokkal megoldani. **[Wil_99]**

7.3. Objektumok és osztályok

Mit tárol egy LDAP adatbázis? Egy címtárban tulajdonképpen különböző entitások vagy objektumok jellemzői kerülnek tárolásra. Ilyen objektumok lehetnek hálózati eszközök, felhasználók, telepített szoftverek. Az objektum jellemzőit attribútumoknak nevezzük **[RFC1778]** Egy objektum egy attribútumának több értéke is lehet, az értékek típuskezelése meglehetősen nagyvonalú (nincs szigorú típusmegkötés az attribútumok esetén). Minden objektum, mely az adatbázisba kerül, rendelkezik egy speciális attribútummal, melynek neve: objektum osztály *(objectclass)*. Ennek az attribútumnak az értékei határozzák meg, hogy az adott objektum mely objektum osztályba tartozik.

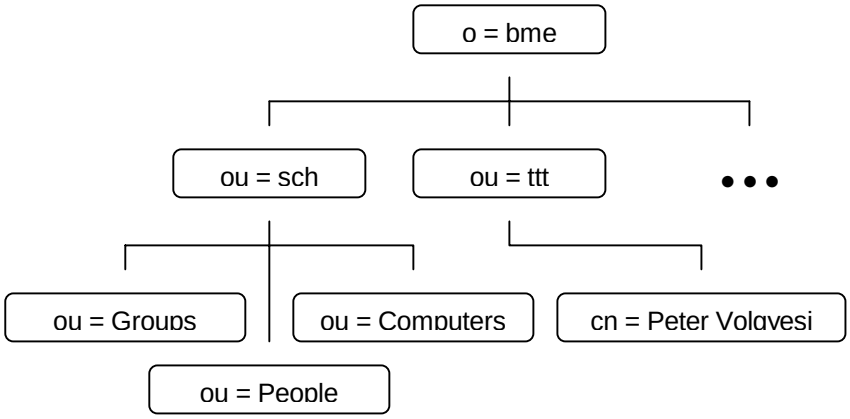
Minden osztály előír kötelező ill. opcionális attribútumokat, így az objektum osztály attribútum értékei meghatározzák, hogy milyen további attribútumokat kell ill. lehet tárolni az adott bejegyzésben. A szabványosnak tekinthető osztályokon és attribútumokon túl bárki definiálhat (a szerver konfigurációja során) új típusokat. Egy személyi rekord például a következőképpen nézhet ki (3. táblázat):

cn=Peter Volgyesi, ou=TTT, o=BME	
objectclass:	top
objectclass:	person
cn:	Peter Volgyesi
Surname:	Volgyesi
title:	student
tel:	+36-1-1234567

3. táblázat Egy tipikus LDAP bejegyzés

7.4. Elemek címzése

Hogyan valósul meg az objektumok címzése? A fenti bejegyzés első sora az ún. egyedi név, *DN (Distinguished Name)*. Ez nem más, mint néhány attribútum-érték pár felsorolása veszővel elválasztva **[RFC1779]** Szabványosított azon attribútumok köre, amelyek felhasználhatók a DN képzésében. Hogy ezek közül melyek lesznek használva, csak azon múlik, s az a egyetlen követelmény, hogy az objektumok valóban egyedi neveket kapjanak. Az attribútumok sorrendje fontos, a címzés segítségével ugyanis hierarchikus struktúrát lehet építeni. A legspecifikusabb attribútum kerül a legbaloldalibb helyre (Ezt egyébként *RDN*-nek hívják). Nincs megkötés arra nézve sem, hogy objektumok csak az így létrehozott fa leveleiben helyezkedhetnek el, sőt, minden egyes csomópontban célszerű elhelyezni egy-egy bejegyzést. Nagyon fontos tudni: az LDAP nem tárolja a DN-t az objektum attribútumaként, így nem is lehet keresni a DN alapján. Ezért célszerű létrehozni a csomópontokban bejegyzéseket, melyeket rendelkeznek az RDN-nel azonos attribútummal. Lássunk egy példát egy ilyen hierarchikus LDAP fára (10. ábra):



10. ábra Egy LDAP fa (DIT) szerkezete

7.5. Műveletek

Milyen műveletek végezhetők egy LDAP szerveren? Meglepően kevés művelet áll rendelkezésünkre. A legfontosabb és leggyakrabban használt funkció a *keresés*, mely lehet teljes (egy adott szinttől lefelé a teljes fában való keresés), vagy egy adott szintre, esetleg egy bejegyzésre korlátozódó **[RFC2254]** . Kezdeményezhetjük új objektum felvételét a címtárba az attribútumai megadásával, ill. módosíthatjuk vagy törölhetjük az objektumot. A bejegyzéseken, részfákon elvégezhető műveletek erősen függhetnek attól, hogy mi engedélyezett a címtár gazdája a keresőm számára. Az azonosítást a címtárak esetén kötésnek *(bind)* hívják. Kötéskor egy DN-t és egy jelszót adunk meg az LDAP szervernek. Az adott DN-hez tartozó bejegyzést megkeresve és annak password attribútumát az általunk megadott jelszóval összevetve az LDAP kiszolgáló meghatározza hozzáférési jogainkat. A viszonylag kevés és egyszerű művelet lehetővé teszi, hogy a kliens oldalon kisméretű egyszerű LDAP API-kra legyen szükség, így a szolgáltatás igénybevételéhez nem kell alapjaiban megrengetni egy – esetleg már létező - alkalmazást.

7.6. Az LDAP lehetőségei a menedzsment terén

Mindezek után nézzük meg, hogy miért alkalmas az LDAP a bevezetőben említett feladatok megoldására !

- Az LDAP valóban gyártófüggetlen, az IETF által szabványosított ill. részben szabványosítás alatt álló protokoll, így valóban betöltheti a menedzsment alkalmazások közötti közös nyelv szerepét ezen a téren.
- Az LDAP hierarchikus tárolási struktúrája megfelelő szerkezetet biztosít egy hálózat és az azon található objektumok tárolásához. (Pl. hálózat → szegmens → gépek → alkalmazások). Így a tárolási és címzési mechanizmus gondoskodik automatikusan az eszközök közötti hierarchia-viszonyok rögzítéséről.
- Az LDAP objektumok alapvető sajátossága az objectclass attribútum általi osztályokba sorolás. Ez a tulajdonság szintén rendkívül alkalmassá teszi az LDAP-t olyan heterogén hálózati struktúrák tárolására, ahol fontos az egyes hálózati elemek megfelelő típusba sorolása. Továbbá az LDAP objektumok által megvalósított ill. tárolt attribútumok ugyanúgy az objektum osztályától függenek, ahogy a hálózati objektumon elvégezhető műveletek az adott eszköz típusától.
- A várható lekérdezések és írási műveletek aránya szintén megfelel annak a követelménynek, amit az LDAP bevezetőjében mondtunk. Ez azt mutatja, hogy valóban címtár szolgáltatásra van szükség ezen hálózatmenedzsment feladat megoldásához.
- Az LDAP protokoll ugyanazokat a BER rutinokat használja, mint az SNMP, így a meglévő hálózatmenedzsment alkalmazásokba az LDAP integrálása egyszerűbb feladat.

8. Fejezet

8. RIASZTÁSOK, KORRELÁCIÓVIZSGÁLAT

Összetettebb, sok objektumot tartalmazó hálózatok felügyelete során rendkívül fontos, hogy a hibák során generálódó riasztások közül képesek legyünk kiszűrni a valóban fontosakat, ill. a hibák gyökerét okozókat. A hálózatmenedzsment eszköz kialakítása során törekedni kell arra, hogy az összefüggések feltérképezését már a számítógép is segítse, és ne árassza el a felhasználót redundáns riasztás-záporral.

Ezt a – nagyon fontos – követelményt támasztja alá az amerikai Three Mile Island-en '79 márciusában bekövetkezett reaktorbaleset, melynek okait az eset után hosszú időn át kutatták (kutatják). Többek között John Kemény (BASIC programozási nyelv) is megbízást kapott a hibák feltárására. Jelentésében **[Kem_79]** arra hívja fel a figyelmet, hogy a meghibásodások és apróbb emberi mulasztások miatt jelentkező problémák valószínűleg orvosolhatók lettek volna, ha a vezérlőteremben ezek ésszerű és nem tömeges riasztásokat okoznak.

8.1. Esemény feldolgozás

Az riasztások forrásai többnyire külső ágensek (pl. SNMP trap-et küldő hálózati berendezések) vagy a menedzsment architektúra monitor jellegű moduljai (pl. folyamatos adatgyűjtés során egy paraméter átlépte a megengedett küszöbértéket). Az események egy központi adatbázisba érkeznek, ahol a köztük levő kapcsolatokat egy külön modul próbálja feltérképezni. A kapcsolatok feltárása során a modul a következő döntéseket hozhatja:

- Riasztás figyelmen kívül hagyása (pl. ismétlődő riasztás)

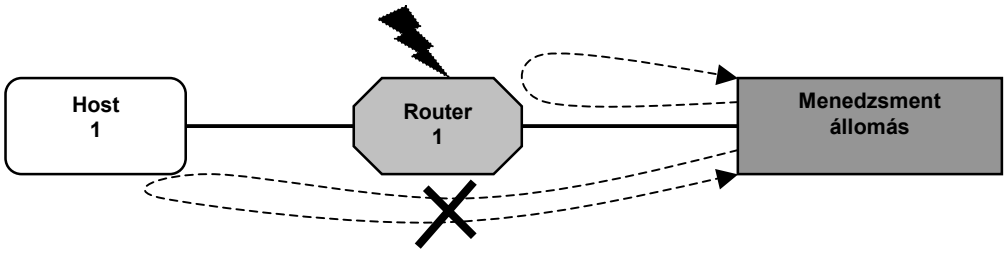
- Riasztások összevonása (pl. egy hálózati eszköz több interfészének közel egy időben történő meghibásodása)

- Új esemény generálása (pl. rendszeres hibajelentés megszakad, és a modul egy idő után úgy dönt, hogy ez a hiba megszűnését jelenti)

8.2. Események közötti összefüggések

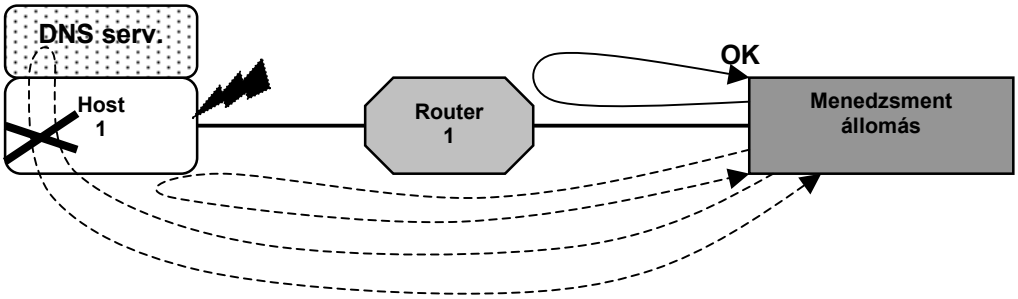
Az eseménykorreláció legnehezebb része természetesen az összefüggések felismerése. Már a triviális esetekben – pl. ugyanannak a riasztásnak az ismétlődése - is sokszor kényszerülünk nem egzakt megoldásokra. Most röviden áttekintjük azokat az összefüggéseket, melyek felismerését viszonylag egyszerűen automatizálhatjuk és a menedzsment szoftver gondjaira bízhatjuk. **[She_94]**

Horizontális függés a menedzselt objektumok között. Ez azt jelenti, hogy bizonyos hálózati elemek elérhetősége – a menedzsment állomás felöl – függ a köztes eszközök működésétől. Egy ilyen függőségi viszony esetén a közelebbi eszköz (pl. router) meghibásodása a távolabbi eszközök elérhetetlenségét, és így felesleges riasztásokat eredményez. Erre látunk példát a 11. ábra



11. ábra Horizontális függés

Vertikális függés a menedzselt erőforrások között. A menedzselt objektumok sokszor nem kizárólag hálózati eszközök, interfészek, hanem magasabb szinten elhelyezkedő alkalmazások is, melyek a hálózat működése szempontjából kritikus jelentőségűek. Ilyen szolgáltatások többek között a DHCP, WINS, DNS szerverek. Egyes esetekben szükség van olyan komplex alkalmazások megfigyelésére is, mint egy adatbázis szerver vagy web kiszolgáló. Ezek elérhetőségét azonban egyértelműen befolyásolja a gazda számítógép elérhetősége. Ha ez a számítógép már a hálózati rétegben nem érhető el (pl. interfész meghibásodása), akkor az alkalmazások elérhetetlenségéről felesleges riasztásokat küldenünk az üzemeltetés felé. A szolgáltatások függését mutatja a 12. ábra



12. ábra Vertikális függés

Ismétlődő jelentések. Az egyik legtriviálisabb felismerés, hogy ugyanarról a hibáról folyamatosan érkező riasztások esetén a további riasztásokat figyelmen kívül hagyjuk, vagy kicsit eltérő tartalmukkal (új időbélyeg, módosított hibaüzenet) aktualizáljuk a felvett eseményt. Az ismétlődő riasztásokat tipikusan olyan komponensek generálják, melyek monitor jellegű megfigyelést végeznek. Az ismétlődések felismerésére klasszikus példa a UNIX rendszerek syslogd nevű folyamata, mely a rendszerüzenetek naplózásáról gondoskodik. Bármennyire is egyszerűnek tűnik azonban ennek a kapcsolatfajtának a felismerése, még itt is vannak bizonytalansági tényezők: pl. két összetartozónak ítélt riasztás között eltelt maximális idő, üzenet szövegének megengedett módosulása, stb.

Események életciklusa. A legtöbb hálózati esemény különböző fázisokon megy keresztül, mely fázisok követése szintén az eseménykorreláció feladata. A hiba megjelenése, fennállása, a hibaelhárítás megkezdése, és a hiba megszűnése mind-mind önálló riasztásokat válthat ki a megfigyelő modulok szintjén. Ezeket a menedzsment rendszernek össze kell tudni vonnia, ill. amennyiben a felhasználó kíváncsi az esemény lefolyására, akkor az összevont eseményt ki kell tudni bontania, megmutatva az esemény teljes életciklusát.

Jóváhagyott események, nem menedzselt objektumok. Lehetnek a hálózaton olyan erőforrások, melyek felügyeletét nem szeretnénk megvalósítani. Ilyen lehet pl. egy olyan hálózati eszköz, melyhez nem adtak hozzáférést, vagy menedzselését mások végzik. (Tipikusan a hálózatok határán lehetnek ilyen berendezések.) Ugyancsak lehetnek olyan események, melyekkel a továbbiakban nem akarunk foglalkozni. Ezeket az eseményeket célszerű lezárni vagy jóváhagyni, így hiába kapunk róluk további riasztásokat, azokat az eseménykorrelációs modul figyelmen kívül fogja hagyni. A nem menedzselt objektumokról érkező üzenetek szintén eldobásra kerülnek.

Előre bejelentett karbantartási munkák. Az előző ponthoz hasonló, de időben az eseményeket megelőző beavatkozást jelent. Az előre tervezett karbantartási munkákat, azok időpontját és az általuk érintett hálózati részeket felvehetjük a korrelációs modul adatbázisába, így a munkák által kiesett erőforrásokról nem jut riasztás a felhasználói felületre.

Különböző protokollok által szerzett információk. A menedzsment rendszerek – ahogy ezt az általam készített kísérleti megvalósítás is jól mutatja – a felügyelet során több különböző technológiát, protokollt „bevetnek" a minél pontosabb információk megszerzése érdekében. Sok esetben egyszerre több úton kapunk ugyanarról a hibáról jelzést (pl. egyszerre használt CMIP, RPC, SNMP). A menedzsment rendszernek ezeket szintén össze kell tudni vonnia, lehetőleg egyszerre felhasználva mindegyik protokoll által szolgáltatott információt.

8.3. Nehezen kezelhető problémák

A fenti pontokon túl természetesen sokkal bonyolultabb egymásra hatások is jelentkeznek egy valódi hálózaton. Ezek egy részét vagy nagyon nehezen lehet automatikusan felismerni, vagy az adott hálózat jellegzetességei miatt egyediek. Most csak két csoportot emelnék ki a nehezen kezelhető összefüggések közül:

- Egy szinten elhelyezkedő szolgáltatások egymásra hatása. Pl. elképzelhető, hogy a DNS szerver leállása a többi alkalmazásban szokatlan jelenségeket produkál. Ugyancsak sok fejtörést okozhat UNIX rendszerekben az ident nevű folyamat leállása, mely a kapcsolatfelvételt zavarhatja meg kellően biztonságosra konfigurált szerverekkel.

- A hálózati topológia sokszor nem olyan egyszerű, mint ahogy azt a horizontális függés során feltételeztük. Egy redundáns összekötéseket tartalmazó és intelligens routing protokollt alkalmazó hálózat esetén már nagyságrendekkel nehezebb az ilyen jellegű összefüggések feltárása.

8.4. Nyitott rendszerek

A helyi sajátosságok kiaknázása végett egyre több menedzsment környezet biztosít felületet saját összefüggések definiálására. Ezek a felületek – a feladat jellegéből adódóan – nem egyszerűek. Az általam látott egyik legügyesebb megoldás a Cisco Info Center terméke, ahol SQL jellegű nyelven lehet megfogalmazni ezeket az összefüggéseket **[Cis_99]** .

9. CÉLKITŰZÉSEK

A kísérleti implementáció – továbbiakban **EC/M** (Event Correlation/Monitor) – elsődleges célja az előző fejezetekben bemutatott elképzelések kipróbálása ill. demonstrálása volt. Az alkalmazás fejlesztésekor ezért nagy hangsúlyt fektettem arra, hogy tetszőleges pontján vizsgálható legyen a rendszer működése. Ezt egyrészt minél több különálló (és külön folyamatként futó) modul segítségével, másrészt egy interpreter alapú nyelv felhasználásával értem el. Így lehetőségem volt a modulok nagyon gyors és könnyű módosítására, belső változók kijelzésére, a teljes alkalmazás megbolygatása, újrafordítása nélkül.

Az alkalmazás a hálózati objektumok monitorozása során riasztásokat generál, melyek központi adatbázisba kerülnek. Külön modul gondoskodik az összefüggések feltárásáról. A felhasználó egy böngésző és az alkalmazásba integrált WEB kiszolgáló segítségével megtekintheti az értelmezett eseményeket, a „nyers" riasztásokat és az objektum címtárat. Mivel az alkalmazás elsődleges célja a lehetőségek kipróbálása volt, ezért a címtár módosításához külön felületet nem kívántam készíteni. E feladatot tetszőleges LDAP klienssel meg lehet oldani. A címtár feltöltésének kezdeti lépéseit segítem egy hálózati felderítő modullal. A WEB felületen lehetőségünk van egyes események jóváhagyására/lezárására.

9.1. Megoldandó problémák, feladatok

Számos lehetőség és információforrás kínálkozik egy menedzselni kívánt hálózaton. Így fontos volt már az elején meghatározni azokat, melyeket az alkalmazás monitorozni fog. E funkciók kiválasztásakor is a kísérleti jelleg

motivált, ezért megpróbáltam minél különbözőbb feladatokat megoldani, s megmutatni az eseménygyűjtésben és kapcsolatfeltárásban rejlő integrációs lehetőségeket. A következő monitor funkciókat választottam:

- *ICMP megfigyelés*: a hálózati eszközök elérhetőségének közvetlen tesztelése (a közismert ping programhoz hasonlóan)

- *DNS szolgáltatás megfigyelése*: az IP hálózati szolgáltatások közül az egyik legkritikusabb a névfeloldásért felelős szerver működése. Ez a monitor ezeket a szervereket ellenőrzi.

- *WEB szolgáltatás megfigyelése*: bár a hálózat működése szempontjából nem kritikus, de az egyik leggyakoribb alkalmazási terület a WEB. Amennyiben publikus szerverről van szó, a szolgáltatás kiemelt fontossággal bírhat.

- *Útvonalválasztók megfigyelése*: elsősorban az SNMP protokoll felhasználásának demonstrálása végett a router-ek hálózati interfészeit monitorozzuk a router oldali ágens segítségével. Az információt felhasználhatjuk teljes szegmensekre vonatkozó következtetések levonására.

- *SNMP Trap-ek fogadása*: ez olyan funkció, melyet minden menedzsment platformnak – legyen az akár csak kísérleti – „illik" megvalósítania. Az eseménykorrelációban ezeket a „kéretlen" információkat azonban nehéz feldolgozni.

9.2. Építőkövek

Számos a megvalósításhoz szükséges külső eszközt – mint amilyen az LDAP címtár, SQL szerver, SNMP protokoll könyvtár – építettem be az alkalmazásba. Ezek kiválasztásakor szempontjaim a lehető legszabadabb licenszpolitika, a nyílt szabványokon alapuló és lehetőségekhez mérten

platformfüggetlen megoldás. Mindegyik felhasznált eszköz kutatási, oktatási célra ingyenes, és forráskódja szabadon hozzáférhető. Az eszközök részletes bemutatása az A. függelékben található.

9.3. Szerkezet

A több – viszonylag független – modulból álló szerkezetnél fontos meghatározni azokat a csatornákat, melyeken keresztül a modulok egymással kommunikálnak. Ezen a téren nem akartam semmiféle önálló protokollt bevezetni, kizárólag a meglévő lehetőségeket alkalmaztam, melyek a következők:

- Parancssori paraméterek és környezeti változók
- Közös konfigurációs file
- LDAP és SQL adatbázisok
- UNIX signal-ok

A sok modul és külső eszköz összehangolt és egyszerű indítását, állapotuk lekérdezését és minél biztosabb leállítását is meg kellett oldani. Ezek parancssorból indítható programok segítségével valósulnak meg.

9.4. Platformfüggetlenség

A megvalósítás során törekedtem arra, hogy minél kevésbé kötődjek azokhoz a platformokhoz, melyeken az alkalmazást fejlesztettem (Linux és Sun Solaris). Bár néhány külső eszköz csak a UNIX rendszereket támogatja, ezek külön-külön kiválthatók más platformokat támogatóval (pl. tetszőleges más LDAP szerver használható az alkalmazással). Ez kifejezetten a moduláris felépítésnek és a modulok közötti nyílt szabványú protokolloknak

köszönhető. Az általam írt kód – kivéve az indításról gondoskodó shell parancsfile-okat – Tcl nyelven készült, s így rendkívül hordozható.

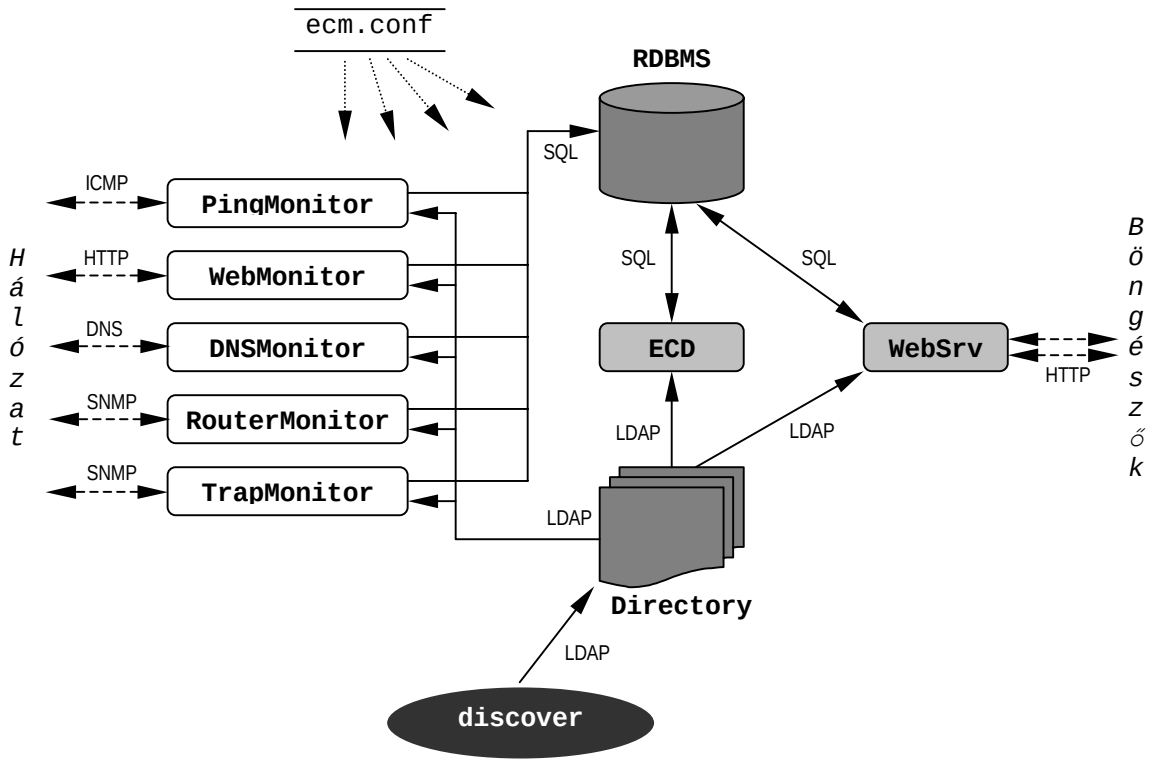
10. A KIFEJLESZTETT FELÜGYELETI ARCHITEKTÚRA BEMUTATÁSA

Ebben a fejezetben az elkészített alkalmazás szerkezetét, működését mutatom be. Először megismerjük az EC/M általános architektúráját, majd az egyes modulok feladatait, vázlatos működését. Néhány segédprogramot is bemutatok.

A hálózati modell és az eseményeket tartalmazó adatbázistáblák bemutatása után ismerkedünk meg a legfontosabb algoritmusokkal.

10.1. Szerkezet

Az EC/M alkalmazás szerkezete látható a 13. ábran. Az általam készített háttérfolyamatokat a lágy sarkú téglalapok mutatják. Az ellipszis alakzat az előtérprogramokat jelöli, míg két vonal között helyezkednek el a konfigurációs állományok. (Minden általam készített modul egyetlen központi konfigurációs állományt használ, melynek neve ecm.conf.) A nyilakon feltüntettem az adott információs csatornán használt protokollt. A rendszer határain a menedzselt hálózat, valamint a böngészők (WWW) állnak. Az ábrán – az átláthatóság növelése érdekében – nem szerepelnek azok az előtérprogramok, melyek a rendszer elindításáról, leállításáról ill. a folyamatok állapotának lekérdezéséről gondoskodnak. Ezekről a kiegészítő programoknál lesz szó.



13. ábra Az EC/M szerkezete

10.2. Az egyes modulok és feladataik

A relációs adatbázis több táblát tárol, melyekbe egyrészt az új riasztások kerülnek a monitor folyamatok révén, másrészt a korreláció során a magasabb szinten értelmezett események íródnak. A táblákról később még lesz szó.

A címtár (LDAP Directory) feladata a hálózati objektumok, mint LDAP bejegyzések tárolása, az objektumok attribútumainak (pl. IP cím) és hierarchikus viszonyainak rögzítése. Az LDAP kiszolgálót a háttérfolyamatok csak lekérdezik, a beírás (az objektum adatbázis módosítása) kézzel vagy a hálózati felderítő programmal lehetséges.

A megfigyelő modulokról és az általuk elvégzett feladatokról már volt szó. Itt annyit érdemes megemlíteni, hogy az egyes modulok bizonyos attribútum megléte esetén veszik fel az általuk menedzselt eszközök közé az adott

objektumot, és mindegyik monitor folyamat közvetlenül éri el mind az LDAP mind az adatbázis szervereket. Az útvonalválasztókat felügyelő modulból ennek a fejezetnek a végén mutatok egy érdekes algoritmust.

Az *ECD* nevű modul az események közötti összefüggések felderítését végzi. A „nyers" riasztások kiolvasása és feldolgozása során a korrelált eseményeket visszaírja az adatbázis megfelelő tábláiba és a feldolgozott riasztásokat megjelöli. Elméleti jelentősége miatt az itt végrehajtott lépésekkel még foglalkozunk.

A *WebSrv* folyamat a böngészők kiszolgálásáért, az alkalmazás felhasználói felületéért felelős. Ez egy nagyon egyszerű – szintén Tcl nyelven megírt – WEB kiszolgáló, mely tartalmazza azokat a kódrészeket, melyek a felhasználói felületet előállítják (így nincs szükség CGI vagy más bonyolultabb és erőforrásigényesebb interfész igénybevételére). A felhasználói felület bizonyos elemeit (pl. menü, súgó) statikus HTML file-okban tárolja a szerver. **[Ball_99]**

10.3. Kiegészítő programok

A felderítést végző segédprogramon túl három fontosabb parancs van, mely a teljes rendszer elindítását *(ecmstart)*, leállítását *(ecmstop)* és a folyamatok jelenlegi állapotának lekérdezését *(ecmstatus)* segíti. A modulok megfigyelése és leállítása a - minden modul által induláskor letárolt - folyamat azonosító (PID) segítségével történik. A modulok állapotát a WEB felületen keresztül is megnézhetjük (feltéve, hogy maga a WEB kiszolgáló még fut).

10.4. Az alkalmazás könyvtárszerkezete

A számos modulból álló alkalmazáshoz megpróbáltam átlátható fizikai (tárolási) szerkezetet kialakítani. Az átláthatóság mellett fontos szempont volt a modulhatárok megőrzése, így az egyes részkomponenseket könnyebb volt külön-külön fejleszteni (és az új verziókat archiválni), ill. jelentősen

egyszerűbb a modulok (pl. LDAP szerver) cseréje. A külső eszközök felhasználása során ezeket a csomagokat külön -saját- könyvtárakba telepítettem.

A modularitás hátulütője, hogy több helyen vannak futtatható állományok, dinamikusan töltődő könyvtárak, kézikönyvek (*man* oldalak). Ennek a problémának az orvoslása egy olyan parancsfile *(sh.rc)* segítségével történik, mely gondoskodik a különböző elérési utak beállításáról. Mivel az EC/M célszerűen külön felhasználói azonosító alatt fut (ecm felhasználói név alatt), ezt a parancsfile-t ajánlatos a belépéskor automatikusan lefutó file-ok közé venni. A beállított környezeti változók:

- *PATH*. A különböző EC/M könyvtárakban levő futtatható file-ok elérési útját fűzi hozzá ehhez a változóhoz

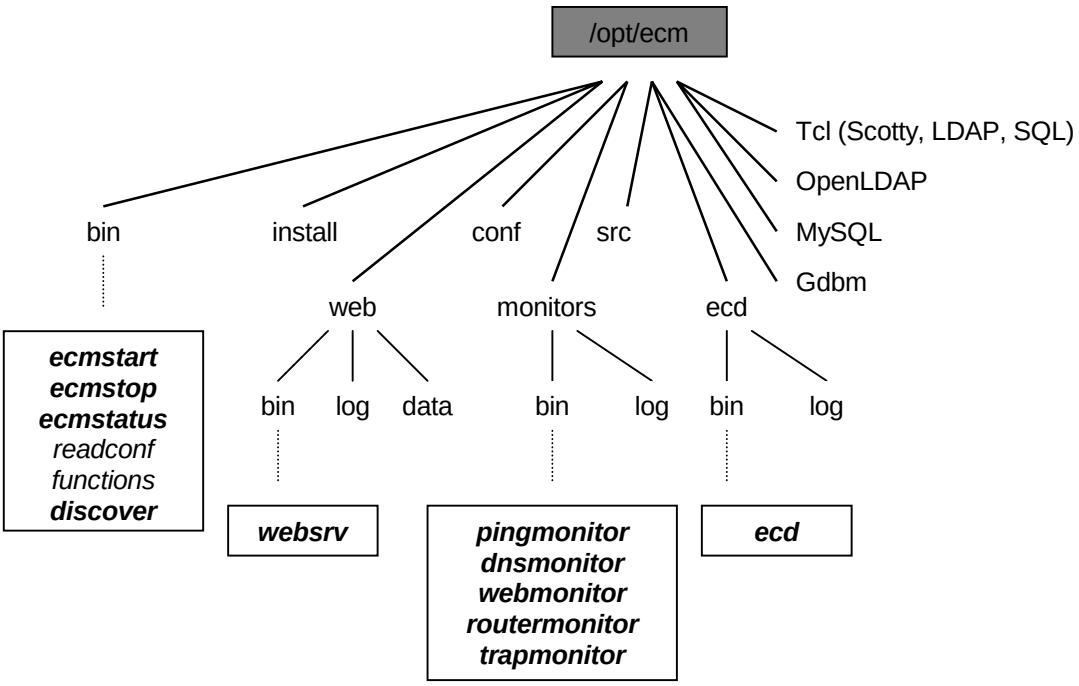
- *LD_LIBRARY_PATH*. Azokon a UNIX rendszereken, melyek az ún. ELF formátumban tárolják a futtatható állományokat – ilyen a Linux és a Solaris – ezen az elérési úton keresi a könyvtárbetöltő program a dinamikusan kapcsolt könyvtárakat (.so kiterjesztésű file-okat). A legtöbb külső modult kénytelen voltam dinamikus könyvtárak alakjában fordítani. Ez nem túl kényelmes a fenti változó állandó beállítása miatt, azonban a Tcl/Tk nyelven használt kiterjesztések csak ebben az esetben töltődhetnek be dinamikusan az interpreterbe. (Például az LDAP vagy az SQL kiterjesztés igényli az OpenLDAP ill. a MySQL csomag dinamikus kliens könyvtárait)

- *MANPATH*. A külső modulok kézikönyv oldalainak elérési útját veszi fel

- *ECDDIR*. Nagyon sok modul és program használja ezt a környezeti változót a többi file eléréséhez, ezért ennek megfelelő beállítása alapvető fontosságú a rendszer megfelelő működéséhez. Az

alkalmazás gyökérkönyvtárát kell, hogy tartalmazza (alapértelmezés: /opt/ecm)

A kialakított könyvtárszerkezetet mutatja a 14. ábra.



14. ábra Az EC/M file szerkezete

Az alkalmazás indításához, leállításához szükséges programok, valamint más parancssorból a felhasználó által indítandó programok az alkalmazás *bin/* könyvtárában vannak. Ugyanitt vannak azok a file-ok melyek hasznos és több modul által használt feljárásgyűjteményeket tartalmaznak.

A WEB kiszolgáló könyvtára a *web/* elérési úton van. Itt három alkönyvtárba szerveztem a futtatható programot, a napló (és folyamatazonosítót tartalmazó ún. *pid*) file-okat, valamint a kiszolgáló statikus adatait: HTML file-okat és képeket.

A megfigyelő modulok hasonló szerkezetű alkönyvtárban vannak a *monitors/* könyvtár alatt. A naplózás itt nincs megvalósítva, hiszen a szükséges

információk az esemény adatbázisba kerülnek. A kritikus hibákat ezek a modulok a szabványos hibakimenetükre küldik.

Az *ecd/* úton található a korrelációs modulhoz tartozó napló és bináris könyvtár. Bár itt sincs naplózási funkció kialakítva, erre egy teljes értékű rendszer esetén szükség lenne.

A modulok által használt konfigurációs paraméterek egyetlen file-ban (ecm.conf) vannak a *conf/* könyvtár alatt. Az egyes paraméterek jelentését a függelék tartalmazza. A konfigurációs file szerkezete egyszerű: megjegyzésekből és kulcs/érték párokból áll, melyeket egyenlőségjel választ el egymástól. Ennek a file-nak a beolvasása és feldolgozása nagyon egyszerűen történik: a bin/ könyvtárban található *readconf* file minden modul elején végrehajtásra kerül. Ez gondoskodik a konfigurációs file-ban található értékadások végrehajtásáról. Az egyes konfigurációs paraméterek a Tcl programokban változókként jelennek meg, melyet a programok többi része felhasznál. A modulok ügyelnek arra, hogy ha nincs a megfelelő paraméter megadva, akkor azt a modul forrásában definiált alapértelmezett értékkel helyetesítsék.

Az *install/* könyvtár alatt olyan file-ok kaptak helyet, melyek az adatbázis (ecm_init.sql) és az LDAP fa (ecm_init.ldif) kezdeti szerkezetét alakítják ki. Ez a rendszer telepítése során hasznos.

A felhasznált programcsomagok forrásai az *src/* úton vannak. Ezek fordítása és installálása a legtöbb UNIX platformon viszonylag egyszerűen elvégezhető, többnyire egy konfigurációs (GNU autoconf) és egy fordítási fázisból áll.

A lefordított külső modulokokat az *openldap/, mysql/, tcl/, gdbm/* könyvtárak tartalmazzák. A gdbm csomagra az LDAP szervernek van szüksége, az EC/M közvetlenül nem használja. A tcl könyvtár alatt bújnak meg a különböző Tcl-hez fordított kiterjesztések.

10.5. LDAP szerkezet, hálózati modell

Az EC/M hálózati modellje a valós hálózati topológia és eszközpark LDAP címtárba képzését jelenti. A modell megalkotása során fontos volt egy olyan szerkezet kialakítása, mely minél jobban segíti az események közötti kapcsolatok felderítését. A viszonylag gyors megvalósíthatóság miatt néhány egyszerűsítő, korlátozó feltételt szabtam:

- A hálózat fa gráf topológiájú, nincsenek benne hurkok, átkötések (így a horizontális függőségek könnyebben kezelhetők és az LDAP címtárba is egyértelműen leképezhető a hálózat)

- A hálózatot a 3. rétegtől (hálózati réteg) kezdem kezelni, ez alatti szegmentálással (pl. switch-ek) nem foglalkozom. (ezeket a rétegeket ugyanis a rendelkezésre álló menedzsment protokollok is mostohán kezelik)

A modellben négy LDAP objektumosztályt vezettem be: *ecmHost, ecmNetwork, ecmService, ecmRouter*, melyek rendre számítógépet, hálózatot, számítógépen futó alkalmazást és útvonalválasztót reprezentálnak. A hálózat ebben az esetben közös router interfészen levő gépek összessége.

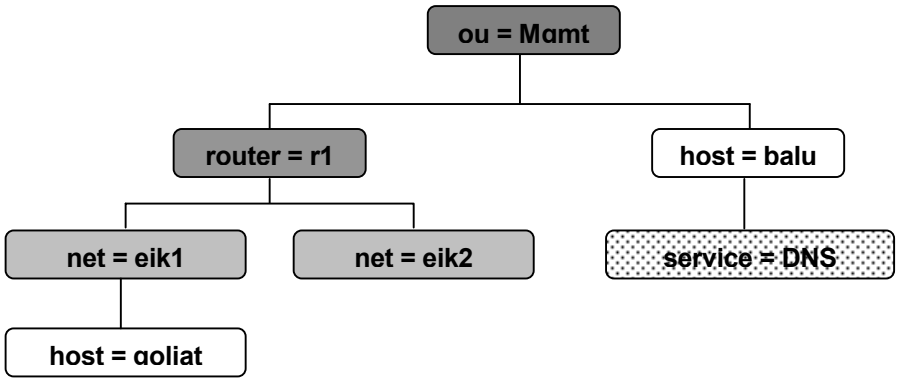
Minden objektum rendelkezik egy *ipaddr* attribútummal, és olyan attribútumokkal, melyek a monitorozás szempontjából fontosak. Az egyes monitor programok által használt attribútumokat tartalmazza a 4. táblázat.

Monitor modul	Felhasznált attribútumok
PingMonitor	*ipaddr* *pingaddr*
DNSMonitor	*ipaddr* *dnszone*
WebMonitor	*ipaddr* *webport*
RouterMonitor	*ipaddr* *snmpcommunity* *routerindex*
TrapMonitor	*trapaddr*

4. táblázat Felhasznált LDAP attribútumok

Az LDAP fa szerkezete az *o=ECM* címkéjű (DN) bejegyzéssel kezdődik, mely alatt a menedzselt hálózatot az *ou=Mgmt, o=ECM* objektum és az alatta elhelyezkedő fa tartalmazza.

A menedzselt hálózatot modellező fában szabályosan váltakozva szerepelnek egymás alatt a hálózatot és a számítógépet, vagy útvonalválasztót jelentő bejegyzések. Tehát ecmHost és ecmRouter osztályú objektum csak ecmNetwork típusú alatt fordulhat elő és fordítva. Természetesen ecmService csak ecmHost alatt megengedett. A fa felépítése a menedzsment rendszer nézőpontjából mutatja a teljes hálózatot, melynek gyökere a menedzsment állomás maga. Egy konkrét részletet látunk a 15. ábran, mely a Schönherz Kollégium hálózatán elhelyezett menedzsment állomás „látószögéből" készült.



15. ábra Példa a menedzselt hálózat modelljére

10.6. Események attribútumai, adatbázistáblák

Az rendszer három adatbázis táblát használ a menedzsment információk tárolására. Az adatbázis elérése közvetlen módon valósul meg minden modulban, lokális gépen UNIX *domain socket*-ek segítségével, míg hálózaton keresztül TCP protokollon.

Az *Log* táblába a nyers, feldolgozatlan riasztásokat helyezik a megfigyelő modulok. A tábla rekordjainak (5. táblázat) legfontosabb attribútumai a riasztás forrása, típusa (melyik modul küldte a riasztást), időbélyege és üzenete. A feldolgozatlan események *processed* attribútuma NULL értékű, és feldolgozás után annak az eseménynek a sorszámát veszi fel, melybe össze lett vonva (idegen kulcs az *Events* táblából).

Attribútum	Jelentése
serial	*esemény sorszáma (auto increment.)*
dn	*esemény forrása*
type	*esemény típusa (monitorra jellemző)*
eventtime	*esemény időpontja*
processed	*feldolgozottság állapota*
message	*monitor által küldött üzenet*
status	*hiba vagy rendben jellegű üzenet*

5. táblázat A Log tábla attribútumai

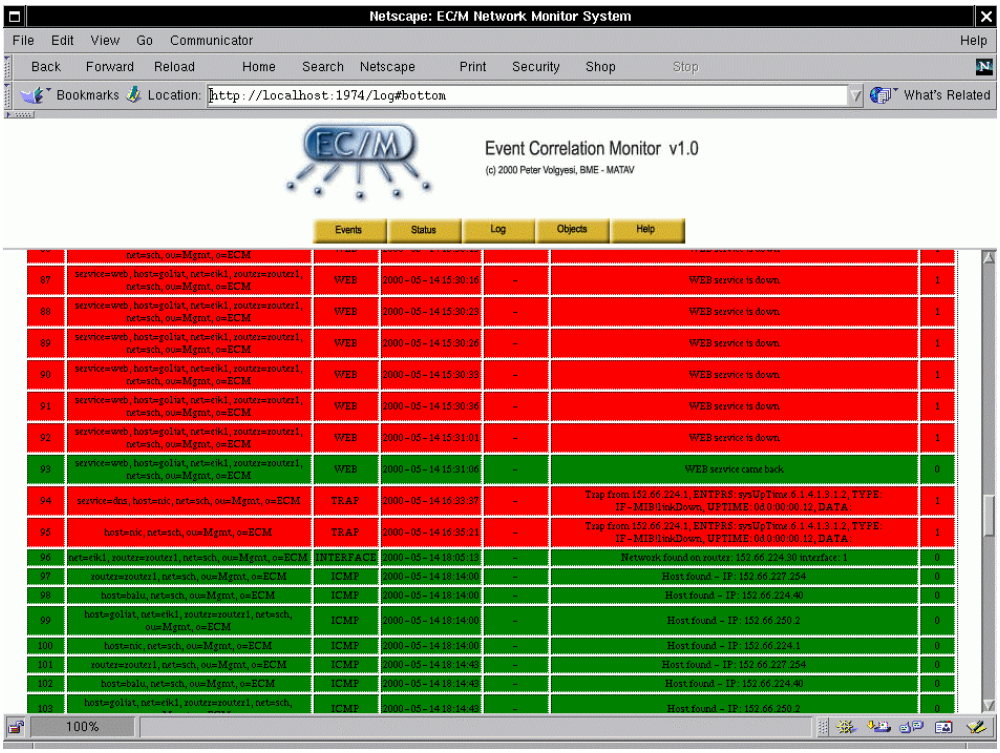
Az eseménykorreláció során a szűrt riasztások ill. a magasabb szintű események az *Events* táblába kerülnek. Ez teljes egészében az ECD modul feladata. Ez a táblázat megegyezik a Log táblával, de rendelkezik egy *closed* attribútummal, mellyel a felhasználó tilthatja az adott eseményhez további nyers riasztások kapcsolását. Természetesen ez a tábla nem rendelkezik a processed mezővel. A korreláció során használt segédtáblázat (*Status* - 6. táblázat) az egyes eszközök jelenlegi állapotát tartalmazza. Ez a táblázat is megjeleníthető a WEB felületen, így rögtön láthatjuk, hogy mely eszközökkel van gond a kérdéses pillanatban.

Attribútum	Jelentése
dn	*esemény forrása*
type	*esemény típusa (monitorra jellemző)*
status	*hiba vagy rendben jellegű üzenet*

6. táblázat A Status tábla attribútumai

10.7. A WEB felület

Az EC/M felhasználói felülete rendkívül könnyen átlátható. A képernyő felső részén található menüsor segítségével váltogathatunk a különböző nézetek között. Az első három menüpont a három adatbázis tábla sorait mutatják meg státuszinformációtól függő színezéssel. Az Events táblából lehetőségünk van az adott eseményben összevont nyers riasztások megtekintésére, valamint adott események lezárására.



16. ábra A Log ablak

Az adatbázis táblák megjelenítésén túl lehetőségünk van az objektum címtár megtekintésére az Objects menü alatt. Adott objektumra kattintva az objektum attribútumait mutatja meg a rendszer.

Az utolsó menüpont rövid tájékoztatást ad a rendszer használatáról.

10.8. Korrelációs lépések

Az események közötti összefüggések automatizált feldolgozása az egyik legnehezebb feladat volt a munka során. A nyers riasztások feldolgozását több fázisban végezzük, az egyes fázisokban felismert összefüggésekhez tartozó riasztásokat eldobjuk. Minden iteráció végén megadott ideig várakozik az ECD modul, majd az ez idő alatt beérkezett friss riasztások feldolgozásával újra végighalad a fázisokon. Egy ilyen iteráció fázisai:

1. A Log táblából kiolvassuk az összes processed = NULL rekordot, ezek a még feldolgozásra váró riasztások

2. Kiolvassuk a Status táblából azokat az eszközneveket, melyek most hibásként vannak megjelölve. Ezekre a továbbiakban azért lesz szükség, hogy az új riasztásokat társíthassuk korábbi problémákhoz.

3. Minden feldolgozandó riasztásra: ha a riasztás DN mezeje valódi része az előző pontban kiolvasott eszközök valamelyikének, és a hiba típusa is egyezik, akkor a riasztást eldobjuk. (A processed mező frissítéséhez ez Events táblából kiolvassuk azt az eseményazonosítót, mely DN-je és típusa a riasztáséval megegyezik és nincs lezárt állapotban.). Ebben a lépésben tehát megszabadulunk azoktól a riasztásoktól, melyek egy – a menedzsment központhoz – „közelebbi" hiba miatt generálódtak. (Horizontális és vertikális függések kiszűrése)

4. Minden még feldolgozandó riasztásra: ha a riasztás DN mezeje és típusa megegyezik a 2. pontban kiolvasott eszközök valamelyikével, akkor a Status táblát frissítjük az új hibaállapottal. Ha az új állapot *OK*, akkor az Events táblából megkeressük azt a rekordot, melynek DN-je és típusa megegyezik a riasztáséval. Ha az új állapot *ERROR*, akkor ugyanezt a keresést hajtjuk végre, de további feltétel, hogy a riasztás és az esemény időpontja között maximálisan csak egy meghatározott idő (*ecddelta* konfigurációs paraméter) telhet el. Ha találunk a fenti kritériumoknak megfelelő rekordot, akkor frissítjük annak időbélyegét, üzenetét ill. hibaállapotát, ellenkező esetben új rekordot viszünk fel az Events táblába. A riasztás processed mezejét ezek után értelemszerűen az új vagy a módosított bejegyzés sorszámára állítjuk be. Itt tulajdonképpen az ismétlődést próbáltuk felfedezni. Vegyük észre, hogy egy régóta hibás állapotot egy új jó jelzés megjavít, míg ha hiba érkezik, akkor azt csak egy bizonyos időintervallumon belül tekintjük azonosnak egy korábbi hibával !

5. Minden maradék riasztásra: ha az új állapot OK, akkor eldobjuk (processed = -1), egyébként mind a Status, mind az Events táblába felvesszük a szükséges információkat, ill. az új esemény azonosítójával frissítjük a processed attribútum értékét. Ebben a lépésbe már olyan riasztások lépnek csak be, melyeket nem tudtunk kapcsolni korábbi eseményekhez. Ekkor is csak a hibás eseményeket érdemes felvenni.

6. A teljes Log táblán „szemétgyűjtés": konfigurációs paraméter *(ecdmaxage)* alapján régi rekordok törlése. Ezzel elkerüljük, hogy az adatbázis túl nagyra nőjön és a WEB felület kezelhetetlenné váljon. Figyelem ! A hibajelzések is elévülnek.

7. Várakozás az *ecdintervall* paraméterrel szabályozott ideig.

Az eseménykorrelációs algoritmus során figyelni kell arra, hogy a köztes lépésekben eszközölt változtatások (adatbázis módosítása, esemény eldobása) a még fel nem dolgozott eseményekre hatással kell, hogy legyenek. A mostani megvalósítás szerint például nem olvassuk újra minden lépésben a Status tábla bejegyzéseit, viszont így ügyelni kell arra, hogy ha ez a lépések során módosul, akkor azt a táblát tartalmazó belső változókban is érvényesíteni kell.

10.9. Egyéb érdekes algoritmusok

Az egyik érdekes feladat az objektum címtár képének előállítása a WEB felületen. Ehhez az LDAP címtárat kellett bejárni, melyhez – a feladat jellegéből adódóan – rekurzív algoritmust használtam. Továbbiakban kihasználtam, hogy az LDAP kereső művelete rendelkezik egy ún. *scope* paraméterrel, melynek megfelelő beállításával a keresést egy adott DN alatt csak egy szinten (és nem a teljes fán) végzi el a kiszolgáló. Az algoritmus így egészen könnyen megvalósítható:

```
bind to LDAP server
LDAPfa(ou=Mgmt, o=ECM)
unbind from LDAP server

proc LDAPfa(dn) {
    entries = LDAPsearch (dn) scope=one
    foreach entry in entries {
        print attributes of entry
        LDAPfa(dn of entry)
    }
}
```

17.ábra LDAP címtár bejárásának algoritmusa

Tapasztalataim szerint a Tcl nyelv ill. az azt megvalósító interpreter a rekurzív algoritmusok esetén sok erőforrást (memóriát) fogyaszt. Nagyobb objektumtárak esetén a fenti algoritmus jelentős veremméretet igényel. A mai Linux és Solaris rendszerek többségén a verem maximális mérete korlátozott – elsősorban a hibás programok kiszűrését segítendő. Előfordulhat (tipikus tünet a WEB kiszolgáló „core dump" típusú elhalálozása), hogy szükség van ennek a korlátnak az emelésére, melyet az *ulimit* paranccsal tehetünk meg.

A megfigyelő modulok közül talán a RouterMonitor a legösszetettebb. Ennek feladata, hogy az útvonalválasztók hálózati interfészeit SNMP protokollal figyelje, és hiba esetén megkeresse, hogy az adott interfészt mely hálózati szegmenshez tartozik – az ehhez szükséges információt a hálózati objektumok routerindex attribútuma tárolja. Az algoritmus egyszerűsített változatát mutatja a 18. ábra.

```
foreach router {
    entries = LDAPsearch (routerindex=*) scope=one
    foreach entry in entries {
        dnindex(router,routerindex) = dn of entry
    }
}

foreach router {
    ifnum = snmp get ifNum.0
    for i = 1 to ifnum {
        astat = snmp get ifAdminStatus.i
        ostat = snmp get ifOperStatus.i
        if ostat != astat {
            alarm from net dnindex(router,i)
        }
    }
}
```

18. ábra RouterMonitor algoritmus

11. A WEB KISZOLGÁLÓ MŰKÖDÉSE

Több megoldást is kipróbáltam a Web szerver kialakításához. Az első elképzelés szerint a rendkívül népszerű Apache szervert használtam, ahol az aktív tartalmat generáló programok CGI felületen illeszkedtek a Web kiszolgálóhoz. Ennek a megoldásnak két nagy problémája volt: az állandóan újrainduló CGI programok nem voltak hatékonyak (minden elindulás során kapcsolódni kellett az RDBMS-hez és az LDAP szerverhez), valamint az Apache szerver egy újabb „C" nyelven írt modult hozott be az egyébként sem kevés külső csomagból álló rendszerbe.

A második próbálkozás az Aol Webserver volt, mely egy rendkívül hatékony Tcl alapú kiszolgáló. Az előbb említett problémák közül az elsőt megoldotta, azonban ez a szerver saját interpreterrel rendelkezik, amibe nem tudtam integrálni a nekem szükséges kiterjesztéseket.

A harmadik próbálkozás a Tcl Webserver nevű eszközzel történt. Ez egy tisztán Tcl nyelven megírt WEB kiszolgáló, így a korábban összerakott interpreterrel is használhattam. Ez a megoldás már tulajdonképpen megfelelt az elvárásaimnak, de rá kellett jönnöm, hogy a program által nyújtott szolgáltatások nagyon kicsi részét használom ki. Bonyolult és terjedelmes szoftver helyett tehát elkezdtem egy szintén Tcl nyelven írt, de minél kisebb, átláthatóbb megoldást találni. Erre – a http protokoll viszonylag primitív konstrukciói alapján – nagy esélyem volt.

A végül felhasznált Web kiszolgáló *Stephen Uhler (Sun Microsystems)* munkáján alapszik, bár jelentős változtatásokat eszközöltem (többek között az aktív tartalom képességével bővítettem).

A WEB szerver két típusú tartalmat képes a böngészők számára előállítani. A statikus tartalmat a szerver adatkönyvtáraiból meríti és ilyenkor file-szolgáltatás jellegű tevékenységet végez csak. Bizonyos URL-ek azonban egy asszociatív tömb indexei, mely indexek segítségével a szerver az adott URL-t kezelő Tcl eljárás nevét kapja. Ha ilyen URL-lel találkozik a szerver, akkor meghívja a megfelelő aljárást az URL „maradék" részét paraméterként átadva. Az eljárás feladata a HTTP header kivételével a teljes tartalom előállítása. Mivel a szerver támogatja az ún. *keepalive* mechanizmust (TCP kapcsolat fenntartása több kérésen keresztül), ezért meg kell várnia az URL-t kiszolgáló eljárás befejeztét a Content-length mező előállításához.

A kiszolgáló a globális információit (pl. port, kapcsolatok száma) egy asszociatív tömbben (Httpd) tárolja. Emellett minden élő kapcsolathoz rendel egy-egy új tömböt (Httpd$socket), mely tartalmazza a kapcsolat azonosítóját, a kiszolgálás állapotát, stb. Indulás után a szerver a konfigurációs file-ban megadott TCP porton figyel, a bejövő kapcsolatokat pedig nem blokkoló módon kezeli. Ez Tcl nyelvben úgy történik, hogy az adott kommunkiációs végpont (socket) írhatósága vagy olvashatósága egy általunk megadott eseménykezelőt aktivizál, mely elvégzi a szükség input/output műveleteket. Ezzel a megközelítéssel elértük, hogy a szerver egy időben több kérést is képes legyen kiszolgálni, ill. az új kérés kiszolgálása megkezdődhet a korábbiak teljes feldolgozása előtt.

A fejlesztés során egy rendkívül bosszantó jelenséggel találkoztam: a WEB kiszolgáló véletlenszerűen – és főleg intenzív használat során – hibásan működött *(core dump)*. A hibát az okozta, hogy a szerver aszinkron módon szolgálja ki a kéréseket, így egyszerre több párhuzamos feladattal is foglalkozik. Sajnos az LDAP kiterjesztés nincs felkészítve arra, hogy egyszerre több párhuzamos kapcsolatot nyissunk, így kénytelen voltam az aktív tartalmat előállító eljárásokat szemaforokkal védeni. (Bár a szemafor megvalósításhoz nem találtam a Tcl-ben atomi test&set műveletet, mégis jelentősen nőtt az alkalmazás megbízhatósága.)

A WEB felület egyik nagy hátránya, hogy nehezen lehet a szerver oldaláról frissítést kezdeményezni. Erre azonban az eseménylistáknál nagy szükség lenne, hogy mindig értesüljünk az új problémákról (a *refresh* gomb folyamatos használata nélkül is). A megoldást a HTML oldalak elején generált META tag adja, mely a böngésző számára előírja a lap ciklikus újratöltését. A következő sor 5mp-es frissítést ír elő:

```
<META HTTP-EQUIV=refresh CONTENT=5>
```

A szerver a főbb tevékenységeket és a működése során jelentkezett hibákat naplózza. Ezt a feladatot a kiszolgáló kódjában a HttpdLog eljárás valósítja meg. Szintén az aszinkron végrehajtás miatt ügyelni kell arra, hogy az egyes kérésekhez tartozó bejegyzések összekeveredhetnek, tehát biztosítani kell, hogy a naplóban követhetők és azonosíthatók legyenek az összetartozó sorok. Ezt a mostani megoldás úgy oldja meg, hogy minden naplóbejegyzés elején feltünteti a kapcsolatot azonosító kommunikációs végpont nevét (socket azonosítót).

12. A MEGFIGYELŐ MODULOK MŰKÖDÉSE

A megfigyelő modulok – feladatuknál fogva – sok hasonlóságot mutatnak. Ebben a rövid fejezetben bemutatom az általános szerkezetüket, valamit a modulok közti apróbb eltéréseket. Jelentősebb különbség a belső algoritmus tekintetében a RouterMonitor megfigyelő modulnál van, erről azonban korábban már volt szó.

A konfigurációs paraméterek beolvasása és feldolgozása után a monitor programok saját folyamat azonosítójukat (PID) rögzítik egy külső file-ban. Erre elsősorban a modul állapotát figyelő ill. leállító parancssori programok miatt van szükség.

A modulok ezután egy rendszeresen – adott időközönként – lefuttatandó eljárást regisztrálnak a Tcl interpreterbe, majd a Tcl eseménykezelőjének adják át a vezérlést. Ettől kezdve a modulok fő végrehajtási szálán már más nem történik. (A kívülről kezdeményezett leállítás – a STOP signal segítségével – innen billenti ki a modult.)

A modulok lelke a rendszeresen végrehajtásra kerülő eljárás. Ennek az eljárásnak a végrehajtása során kapcsolódik a modul az adatbázis és címtár szerverekhez. Ennek hátrányos vonatkozása, hogy minden ciklus elején újra és újra felépülnek a kapcsolatok: Ezek a modulok azonban hosszú időn keresztül futnak, és sokkal megbízhatóbb ha rendszeresen újrakezdjük a kapcsolatokat, mint ha – akár több napon keresztül – feltételezzük, hogy a kapcsolat nem szakadt meg.

A címtár szerverhez történő kapcsolódás után a modulok lekérdezik az összes megfigyelendő bejegyzés szükséges attribútumát és nevét (DN), majd ezeket az adatokat asszociatív tömbökben tárolja.

Az asszociatív tömb elemein végiglépkedve ellenőrzi a megfigyelt objektumok elérhetőségét. Amennyiben hibát tapasztal, ezt egy globális tömbben jegyzi és riasztás rekordot vezet fel az adatbázis szerveren.

Ha normális választ kap, akkor a globális tömb alapján megvizsgálja, hogy a legutóbbi ciklusban volt-e hiba ezzel az eszközzel. Csak abban az esetben generálunk riasztást, ha hibás állapotból váltunk. A globális tömb megfelelő elemét „rendben" állapotúra változtatjuk.

A modulok a szolgáltatásokat ellenőrző ciklus során csak egy SQL sztringet állítanak elő. Az összes eszköz lekérdezése után kapcsolódunk az adatbázis szerverhez, és küldjük es az apránként összerakott utasítást. Ezzel jelentősen csökkentjük az adatbázis szerver és a hálózat terhelését.

Eltérés van az ICMP alapú PingMonitor esetében, ahol a megfigyelt objektumok felé egyetlen lépésben küldünk kérést, és a válaszokat dolgozzuk fel a fent ismertetett ciklus szerint.

Az SNMP Trap üzeneteket fogadó TrapMonitor szintén kilóg a sorból. Itt nincs folyamatos lekérdezés, hanem eseménykezelő eljárást regisztrálunk a beérkező trap üzenetek feldolgozására. Ez az eljárás a beérkező üzenet mezőit kiolvasva megkeresi a riasztást küldő eszköz bejegyzését az LDAP címtárban, majd a riasztás mezőit és az eszköz nevét a riasztásokat tartalmazó táblába írja. Egyelőre minden SNMP Trap üzenetet hibaként regisztrál ez a modul, holott ilyen üzenetek más céllal is jöhetnek. Ezek felismerése azonban több munkát és a trap üzenetek alaposabb ismeretét igényli az alkalmazásban.

13. HÁLÓZATI FELDERÍTÉS

Sok eszközt tartalmazó hálózat objektumainak felvitele egy címtárba nem kis feladat. Ennek az adatbázisnak a karbantartása, összevetése a mindenkori valós hálózattal szintén sok munkát igényel. Ezért minden robosztusabb menedzsment platform rendelkezik hálózati felderítés funkcióval, mely ezt a feladatot próbálja automatizálni. A feladat nem egyszerű, mivel egzakt megoldást aligha lehet találni (ha figyelembe esszük, hogy gyakorlatban nem minden hálózati eszköz menedzselhető pl. SNMP-vel).

Az EC/M-hez kifejlesztett parancssori program erre a problémára keresi a választ. Számos korlátja ellenére – melyeket a fejezet végén foglalok össze, meglepően hasznos információkhoz jutottam használata során.

Mivel nem képes az eszközök teljes mértékű felismerésére, valamint a rafináltabb topológiai viszonyokat is nehezen kezeli, ezért a *discover* program eredményeit még ki kell egészíteni, néha javítani. A program nem közvetlenül az LDAP címtárba dolgozik, hanem egy LDIF formájú kimenetet generál, melyet – akár szövegfeldolgozó eszközökkel: perl, tcl, stb. – pontosíthatunk, majd az *ldapadd* program bemenetére adva az LDAP címtárba tölthetünk.

Az LDIF file-okat előszeretettel használják olyan esetekben, ahol különböző címtárak között kell megvalósítani az átjárást. Például a legtöbb levelező kliensprogram képes a címeket tartalmazó adatbázis exportálására LDIF formátumban.

Az LDIF file-ok szerkezete egyszerű. Az egyes bejegyzéseket egy-egy üres sor választja el egymástól. Minden bejegyzés az adott bejegyzés nevével (DN) kezdődik, melyet az attribútum/érték párok követnek kettősponttal

elválasztva. Minden értékadás külön sorba kerül. Példa egy hálózati szegmenst jelölő LDIF bejegyzésre:

```
dn: net=sch, ou=Mgmt, o=ECM
net: sch
description: Schonherz Zoltan Kollegium LAN
objectclass: ecmNetwork
ipaddr: 152.66.224.0
netmask: 255.255.252.0
routerindex: 1
```

Az LDIF forma ennél bonyolultabb konstrukciókkal is rendelkezik, melyek a bejegyzések utólagos módosítását, törlését szolgálják. Ezekre azonban a hálózati felderítés során nem lesz szükségünk.

A hálózati felderítés lépései:

1. A paraméterként kapott hálózaton (pl. 152.66.224) minden IP cím felé ICMP ECHO kérés küldése. Azokat a címeket, melyek válaszolnak az *activeips* listába tesszük. Ha paraméterként nem „C" oszályú hálózatot kaptunk, akkor rekurzív módon történik a fenti eljárás. Itt a C osztályú lebontásra azért van szükség, mert a program erre a hálózatra egyidőben próbálja kiküldeni az ICMP csomagokat. Ez egy C osztályúnál nagyobb hálózaton hirtelen nagy forgalmat tud generálni *(ping storm),* ami egyrészt a hálózat más szolgáltatásait zavarja, másrészt értékes válaszok veszhetnek el, mivel a választ küldő állomások a hálózati hozzáférésért versengenek.

2. Ciklust kezdünk az 1. pontban megismert IP címekkel, amíg minden fel nem dolgozzuk. Minden iteráció során a TTL mező értékét növeljük a csomagokban. Ennek a módszernek a „tudományos" neve Van Jacobson algoritmus, és a traceroute nevű közkedvelt eszköz is ezen alapszik. Felhasználja, hogy a TCP/IP hálózatokon kötelező a csomagokat továbbító útvonalválasztóknak a TTL mező értékét (legalább) eggyel csökkenteni, valamint a 0-ra futott érték esetén a csomagot eldobni és erről ICMP hibaüzenetet küldeni a feladónak.

Erre elsősorban azért van szükség, mert az útvonalválasztás során kialakulhatnak hurkok (a router-ek autonóm módon választják meg a következő csomópontot), és a hurokba került csomagokat ezzel a módszerrel fel lehet ismerni (anélkül, hogy az útvonalválasztók bármilyen információt tárolnának a továbbított csomagokról !)

2.1. Magas UDP portra csomag küldése a TTL mező beállításával. Várhatóan vagy egy router-től vagy magától a cél eszköztől kapunk ICMP választ. (A cél eszköz azért küldi a választ, hogy jelezze, az UDP port használaton kívül van.) Kikapcsolt, vagy tűzfallal védett gépektől nem fogunk választ kapni.

2.2. A válaszok forrásaitól ICMP MASK kéréssel az interfész hálózati maszkjának lekérdezése. Ha a kérésre nem válaszol az eszköz, akkor netmask = 255.255.255.255. Ezzel megtudhatjuk,. hogy a kérdéses interfész mekkora (al)hálózaton van.

2.3. Ha nem jött válasz a 2.1. pontban, akkor IP cím kivétele a feladatlistából.

2.4. Ha jött válasz a 2.1. pontban és az megegyezik a keresett IP címmel, akkor a gép hálózati címének kiszámítása (IP cím & netmask), és az ehhez tartozó hálózat megkeresése történik egy asszociatív tömbben.

2.4.1. Ha nincs még ilyen hálózat, akkor felvesszük és a kimeneten a hálózathoz tartózó LDAP bejegyzést megjelenítjük.

2.4.2. A keresett IP címett és hálózatot hozzáfűzzük a keresett IP címhez tartozó útvonalhoz, a (host típusú) bejegyzést kiírjuk LDIF formában, és az IP címet kivesszük a feladatsorból.

2.5. Ha jött válasz a 2.1. pontban, de az nem egyezik meg a keresett IP címmel, akkor szintén kiszámítjuk a hálózati címet, és megkeressük az ehhez tartozó hálózatot az asszociatív tömbben. Az forrás IP cím alapján a router-ekre vonatkozó asszociatív tömbben is keresünk.

2.5.1. Ha nincs meg a keresett hálózat, akkor felvesszük és a kimeneten a hálózathoz tartózó LDAP bejegyzést megjelenítjük

2.5.2. Ha nincs meg a keresett router, akkor felvesszük és a kimeneten a router-hez tartózó LDAP bejegyzést megjelenítjük

2.5.3. Ha a forrás IP címe megegyezik az eredeti IP címhez tartozó útvonallista utolsó elemével, akkor az eredeti címhez tartozó bejegyzést kiírjuk, a feladatot eltávolítjuk. (Ez egy apró buktató volt: a router „távoli, nem felénk néző" IP címére küldött csomagokra az eszköz a hozzánk közelebbi interfész címével válaszol.)

2.5.4. A forrás IP címet, és a hálózatot hozzáfűzzük a keresett IP címhez tartozó útvonalhoz

13.1. Kotlátok és néhány buktató

A felderítés során nem vizsgáljuk az eszközökön futó szolgáltatásokat, valamint nem foglalkozunk az SNMP menedzselhetőségükkel. Ezeket az információkat a kimeneti LDIF file-ban kell kézzel beállítani. A hálózati objektumok routerindex attribútuma szintén hiányzik a felderítés után.

Sajnos, nagyon sok operációs rendszer hálózati rétege nem válaszol az icmp mask kérésekre. Ezekben az esetekben feltételezzük, hogy a netmask értéke

255.255.255.255 (lásd: algoritmus leírása). Ez azonban a felismert hálózat erős szegmentálódásához vezet, ami nagyon eltérő lehet a valós helyzettől.

Mivel a EC/M modellje csak fa topológiájú hálózatokat kezel, ezért a felderítés során is ez előfeltétel. Összetettebb hálózatok felismerése jelentősebb több lépést és alaposabb vizsgálatot igényel. **[Sch_93]**

Szintén a modell miatt csak 3. rétegbeli topológia felderítésével foglalkoztam. Számos SNMP menedzselhető 2. és 1. szintben dolgozó eszköz van ma már, és ezekhez a rétegekhez is készültek szabványos MIB-ek, ezért a komoly menedzsment platformok ezeket a kapcsolatokat is feltárják. **[HP_00]**

Az algoritmus végrehajtásakor feltételezzük, hogy a hálózati címek egyediek, vagyis nincs két különböző olyan szegmens a hálózaton, melyek címe megegyezik. Ez „normális" hálózatok esetén törvényszerű kell, hogy legyen.

14. AZ EREDMÉNYEK ÉRTÉKELÉSE

Ebben a fejezetben próbálom meg értékelni a kifejlesztett eszköz alkalmazhatóságát, és az eredeti elképzelések, új ötletek használhatóságát a menedzsment terén. Néhány tesztfeladat bemutatása után a pozitív eredményekkel, majd a felismert hiányosságokkal foglalkozom.

14.1. Tesztek

A tesztelés során elsősorban a korrelált eseményekre voltam kíváncsi, mivel ezek előállításában minden modul részt vesz. Többek között a következő feladatokat sikerült megoldania a rendszernek.

- hálózati interfész lekapcsolása a menedzsment állomáson: ebben az esetben a korrelált események között csak a lokális háló elérhetetlenségéről érkeztek üzenetek. Természetesen a riasztásokat tartalmazó tábla nagyon gyorsan kezdett növekedni, de ezt a korrelációs modul elrejtette előlünk

- „default route" lekonfigurálása: csak a router elérhetetlenségét mutatta az eszköz. Az eseménylistában a routerre vonatkozó hibasorra kattintva megjelentek a mögöttes riasztások is. Meg kell említenem, hogy a szolgáltatások elérhetetlenségét tesztelő WebMonitor sokkal lassabban veszi észre a szolgáltatás kiesését. Ennek okát a http protokoll alatti TCP rétegben keresem, mely az elveszett csomagokat, meg nem kapott válaszokat egy ideig megpróbálja saját hatáskörén belül megoldani.

- DNS szolgáltatás leállítása, majd újraindítása során a két riasztást összevonta a rendszer, és azt szétbontva láthatóvá vált a két eredeti üzenet.

- Azonos – parancssorból küldött – SNMP Trap-eket a rendszer összevont, a többszörözött riasztások csak a hozzájuk tartozó összevont esemény szétbontásával jelentek meg.

14.2. Eredmények, következtetések

Az eredmények alapján úgy ítélem meg, hogy az eredeti elképzelések nagyon jól használhatóak és megvalósíthatóak a felügyeleti alkalmazásokban. A modell jóságának tulajdonítom, hogy már viszonylag hamar és komolyabb buktatók nélkül sikerült az alkalmazás első verziójának az elkészítése. Egyrészt az LDAP felhasználása könnyítette meg a dolgomat (pl. a korrelációs algoritmusnál), másrészt a modulásris szerkezet (a megfigyelő modulok teljesen függetlenek egymástól).

Az egyik legjelentősebb eredménynek tartom, hogy rendkívül skálázható architektúrát sikerült alkotni. Az alkalmazás különösebb módosítása nélkül az egyes modulok különböző gépeken futhatnak. Lehetőség van a monitor modulok kihelyezésére is távoli hálózatok megfigyelése céljából (elosztott menedzsment felé mutat).

Az alkalmazás megbízhatóságát vizsgálva, szinte minden komponens működhet *HA (High Availability)* konfigurációban: az adatbázis szerverek és LDAP szerverek támogatják a replikálás műveletet, a WEB szerverből és monitorokból több futhat. Egyedül a korrelációs folyamat redundáns konfigurációja problémás, bár ennek kiesése nem jár információvesztéssel.

Az LDAP segítségével kialakított hálózati modell egyszerre támogatja a vertikális és horizontális összefüggéseket, mely ismét a korrelációs

tevékenységet egyszerűsítette. (A típus mező inkább informális jellegű az esemény rekordok esetén.)

14.3. Hiányosságok, problémák

Az alkalmazás jelenlegi állapotában nem foglalkozik a menedzsment felület hozzáférhetőségének védelméről és az információk titkosításáról. Bár ezek alapkritériumok egy ipari környezetben használt menedzsment eszköznél, a mostani megvalósításban nem láttam ennek szükségét.

Nehéz kérdés volt annak eldöntése, hogy a monitor modulok közvetlenül az LDAP és SQL szerverekkel vagy egy köztes modulok keresztül kommunikáljanak. Utóbbi esetben a korrelációt már az első beíráskor el lehet végezni, viszont az alkalmazás megbízhatóságát csökkenti *(Single Point od Failure)*. Végül az első utat választottam – többek között az egyszerűbb megvalósíthatósága miatt.

A fejlesztés során rá kellett jönnöm, hogy az események szűrését már a monitor modulokban meg kell kezdeni. Így ezek a modulok a nem hibát tartalmazó üzeneteket csak egyszer küldik el az adatbázisba. Ezzel a forgalmat és az adatbázis méretét jelentősen csökkentettem, viszont a modulokban tárolni kényszerültem a megfigyelt objektumok állapotát.

A szigorú fa gráf hálózati modell nem fogadható el egy valódi menedzsment környezetben. Az LDAP címtárban megoldhatók a „link jellegű" referenciák elhelyezése, így a modell tovább javítható. Azonban ennek kezeléséhez a korrelációs és felderítő modulok jelentős bővítésére van szükség.

Mivel az eseménykorreláció során nincs ismételt feldolgozás (a feldolgozott eseményeket nem vesszük soha többé elő), ezért ha előbb érkezik egy távoli objektumról riasztás, azt nem képes összevonni a rendszer egy közelebbi objektumról – de később – érkező riasztással. Az algoritmus másik gyengéje,

hogy a távoli eszközről érkező OK típusú üzenet nem módosítja a távoli eszközig tartó útvonal köztes eszközeinek állapotát.

A fenti problémák többsége nem az architektúra és felhasznált technológiák miatt jelentkezett. Ezeket több munkával, és bonyolultabb algoritmusokkal meg lehet oldani. Az elképzeléseim kipróbálásához azonban erre nem volt szükség.

14.4. Összegzés

Mind a problémákat, hiányosságokat, mind az elért eredményeket figyelembe véve azt gondolom, hogy a fenti szerkezettel egészen komoly és a gyakorlati életben is használható eszközt lehet készíteni. Alapvető feladatok ellátására az alkalmazás jelenlegi formájában is képes.

A most elkészült alkalmazás használhatóságától függetlenül a MATÁV PKI laborjában végzett másfél éves munka során sikerült megismerkednem a hálózatmenedzsment számos kérdésével és módszerével. Szerencsére több kitérőt és határterületeket érintő kérdéssel foglalkozhattam itt (pl. hálózati csomagok előállítása, protokoll analizátor eszközök, CMIP programozói felület Solaris operációs környezetben, Tcl nyelven fejlesztett MIB böngésző). Az itt szerzett ismeretek összegzése ez a diplomadolgozat, az elkészült alkalmazással pedig igazolni szerettem volna, hogy egyrészt képes vagyok a megszerzett ismeretek innovatív felhasználására, másrészt saját elképzeléseim használhatóak és a gyakorlatban alkalmazhatóak a hálózat felügyelet terén.

KÖSZÖNETNYÍLVÁNÍTÁS

Ezúton szeretném témavezetőmnek, dr. Miskolczi Jánosnak a munkám során nyújtott segítségét és végtelen türelmét megköszönni, rendszeres konzultációink folyamatos munkára ösztönöztek. Korábbi dolgozataim, beszámolóim véleményezése, kritikái szintén sokat segítettek ennek a diplomamunkának az elkészítésében, ezekért köszönettel tartozom egyetemi konzulensemnek, dr. Ziegler Gábornak. A gyakorlati munkák nagy részéhez a MATÁV-PKI eszközeit, szoftvereit használhattam. Végül szeretnék köszönetet mondani dr. Bakay Árpádnak, aki először mutatta meg a Tcl nyelv és a hálózatmenedzsment rejtelmeit.

A. AZ ALKALMAZÁS TECHNIKAI HÁTTERE

A kísérleti megvalósítás technikai részleteivel foglalkozik ez a függelék. Egyrészt a környezet ismertetése a célom, másrészt a teljesség igénye nélkül az érdekesebb implementációs vonatkozásokra hívnám fel a figyelmet. A teljes forrás ill. a felhasznált – nyílt forrású – eszközök megtalálhatóak a mellékelt CDROM-on.

A.1. A felhasznált elemekről

A most következő bekezdésekben ismertetem azokat az eszközöket, melyeket az implementáció során felhasználtam ill. értékesnek ítélek hasonló problémák megoldásához. Alapvető elemeket részletesebben, másokat említés szintjén fogok ismertetni.

A.1.1. Tcl/TK

A Tcl nyelvet *John Ousterhout* fejlesztette ki **[Out_94]** , azzal a céllal, hogy egy magas szintű és könnyen programozható nyelven kösse össze az (elsősorban) C nyelven megírt moduljait. Ezzel egy olyan szkript nyelvet alkotott, mely szintaktikai elemeit tekintve valahol a shell programok és a "valódi" programozási nyelvek *(Perl, Java)* között foglal helyet, amit *Tool Command Language* névre keresztelt.

A nyelv megalkotásakor a cél egy "ragasztó" eszköz elkészítése volt, amiben hatékonyan lehet megírni az alkalmazások szerkezetét, könnyen lehet módosítani azt (mivel az interpreter miatt nem kell mindig újrafordítani a forrást). Az a tény, hogy azóta születtek komoly, több ezer soros alkalmazások

is ezzel az eszközzel, azt bizonyítja, hogy a nyelv önmagában is megállja a helyet, de az elsődleges cél nem ez volt.

A Tcl nyelv sztring középpontú, ami azt jelenti, hogy minden adatot sztringként kezel maga az interpreter. Sőt, az adatok és a program között sem tesz éles különbséget: az, hogy mi értelmeződik adatként és mi programként, valójában szövegkörnyezet kérdése. Ez nagyfokú szabadságot ad a programozónak, viszont korlátozza a komoly matematikai feladatok pusztán Tcl nyelven történő megoldását. (De nem is erre találták ki.) A Tcl támogatja az asszociatív tömbök kezelését ill. hatékonyan kezeli a listákat (amit tulajdonképpen szóközökkel elválasztott elemekből álló sztringekként ábrázol).

A Tcl interpreter eredetileg UNIX platformra készült, azóta sikeresen átültették Windows és Macintosh rendszerekre is, elkészült az interpreter Java alapú megvalósítása ill. egy Netscape plug-in segítségével akár Tcl-ben írt appletekkel (Tclet) is készíthetünk aktív WWW oldalakat. A Tcl-ben megírt programok rendkívül jól hordozhatóak. Az interpreter ingyenes, szabadon használható bármely platformon, és a forrása is bárki számára elérhető.

A Tcl interpreter tulajdonképpen egy programkönyvtár (shared library, dll), melyet saját programjainkhoz kapcsolhatunk, így programunk egyrészt képes lesz Tcl szkriptek értelmezésére, másrészt a Tcl könyvtár beépített parancsait kiegészíthetjük a programunk egyes függvényeivel, melyek az interpreterben parancsokként fognak megjelenni. Így készültek a Tcl könyvtárral egy csomagban lévõ tclsh és wish programok, amelyek tulajdonképpen csak a vázát (pl. main függvény) adják a programnak ill. gondoskodnak a felhasználó és a Tcl könyvtár közötti egyszerű parancssori felületről.

Az Tk nem más, mint a Tcl-hez készített kiterjesztés, mely grafikus képességekkel ruházza fel a Tcl-t. A Tk is egy könyvtár, és a Tcl-hez készített interfészétől eltekintve egy hagyományos Toolkit könyvtárnak tekinthető (amilyen az *Athena*, a *Motif* vagy a *GTK*). A Tk eredetileg az X11 rendszerhez

készült, de ezt is sikeresen átvitték Windows és Macintosh ill. Java platformokra. Fontos tulajdonsága, hogy minden platformon a "natív" GUI elemeit használja, így egy Windows-os Tcl/Tk szkript pontosan úgy fest, mint egy hagyományos alkalmazás. **[Wel_99]**

A Tcl/Tk megjelenése óta (1988) meghódította a világot, és hasonló népszerűségre tett szert, mint a Perl nyelv. Míg a Perl elsősorban a szövegfeldolgozás, rendszeradminisztráció, ill. a CGI programozás területén terjedt el, a Tcl/Tk-t előszeretettel használják grafikus feladatokban, a hálózatmenedzsment területén ill. speciálisabb célfeladatot megvalósító alkalmazások vázaként.

Egy Tcl program újsor vagy pontosvessző karakterekkel elválasztott parancssorokból áll. Megjegyzéseket a kettős kereszt (#) jel segítségével helyezhetünk el, de csak olyan helyre, ahol parancs kezdődne. Egy parancssor a parancs nevéből és tetszőleges szóközzel vagy tabulátorral elválasztott paraméterből áll. Fontos tulajdonsága a Tcl interpreternek, hogy a paramétereknek semmilyen jelentőséget nem tulajdonít, ez a parancsok feladata. (Például az általunk írt parancsok akár futás közben is megváltoztathatják az egyes paraméterek értelmezését.)

Mielőtt a parancsot (ill. a parancsot megvalósító függvényt) a Tcl interpreter meghívná, két fontos műveletet hajt végre a parancssoron. Az első a csoportosítás, ami során meghatározza a parancssor önálló elemeit. Ez általában a szóközök ill. tabulátorok felismerését jelenti, amit felülbírálhatunk kapcsos zárójelekkel vagy idézőjelekkel. Ezek közé tett sztringekben ugyanis a szóközök és tabulátorok nem fognak elválasztójelként működni. A két csoportosító jel közötti különbséget is mindjárt látni fogjuk.

A másik fontos lépés a helyettesítés. Ennek három fajtája van: a változók helyettesítése, mely során a változó értéke kerül a változó helyére (ezt a változó neve elé tett $ jellel érjük el, akárcsak a shell programozásban), a parancsok helyettesítése, amit a szögletes zárójelekkel végzünk (ekkor a

szögletes zárójelek között szereplő parancs eredménye fog a zárójeles kifejezés helyére kerülni) ill. az ún. backslash helyettesítés, amit a C nyelvből is jól ismerünk (pl. \n). A parancshelyettesítés tetszőleges mélységben egymásba ágyazható (rekurzív). A helyettesítést egyébként csak egyszer hajtja végre a parancssoron az interpreter.

A legtöbb beépített parancs valamelyik családhoz tartozik. Ezt úgy kell elképzelni, hogy maga a parancs neve a családnév lesz, azon belül pedig az első paraméter jelöli ki a valódi funkciót. Pl. a file családban a file-műveletekkel kapcsolatos parancsok vannak. Az egyik ilyen parancs: file delete <pathname>, amivel egy file-t törölhetünk. Hasonlóképpen léteznek, pl. a sztringek, a tömbök kezelésre alkalmas parancsok.

Az egyik legfontosabb tulajdonsága a Tcl-nek, hogy rendkívül könnyen ruházhatjuk fel új parancsokkal. Ilyenkor tulajdonképpen egy C függvényt írunk meg, mely paramétereit az interpretertõl kapja olyan formában, ahogy a hagyományos *main()* függvény is kapja a parancssori paramétereket *(int argc, char *argv[])*. A függvény megírásán túl csak annyit kell tennünk, hogy regisztráljuk a Tcl könyvtár egy függvényével az új parancsot, átadva a függvényünkre mutató pointert ill. az általa megvalósított új parancs nevét.

A beágyazhatóságot több célra is fel lehet használni. Egyrészt megírhatjuk az egész alkalmazás vázát Tcl nyelven, és csak a speciális célfeladatokat implementáljuk C nyelven, de használhatjuk arra is, hogy egy rugalmasabb konfigurációs felületet biztosítsunk a felhasználók számára (a programunkból az értelmezett Tcl szkript változóit is le tudjuk kérdezni ill. beállítani).

A.1.2. Scotty

A *Scotty* szoftvercsomagot *Jürgen Schönwälder* fejlesztette ki a Braunschweigi Műszaki Egyetemen. A csomag segítségével lehetőség nyílik rendkívül hordozható hálózatmenedzsment alkalmazások fejlesztésére. Jelenleg egy stabil és egy fejlesztői változata van. A fejlesztői változat előnyei többek között: Tcl/Tk 8.0-ba integrálhatóság, Win32 platform támogatása.

A csomag alapvetően két részből áll: a *Tnm* kiterjesztésből, mely a Tcl/Tk-t menedzsment funkciókkal gazdagítja és a *Tkined* alkalmazásból, mely a Tnm-re épülő hálózatmenedzsment eszköz.

A Tnm kiterjesztés **[Zelt_98]** kétféleképpen vehető igénybe:

- A *scotty* névre hallgató shell-ből. Ez egy tcl shell a Tnm kiterjesztéssel összeépítve
- Tetszőleges Tcl/Tk shell-ből dinamikus linkeléssel: *package require Tnm*

A kiterjesztés a következő protokollokat támogatja:

- *SNMP* (SNMPv1, SNMPv2c, SNMPv2u, MIB file - ASN.1 - értelmezővel)
- *ICMP* (echo, mask, timestamp, udp/traceroute támogatás)
- *DNS* (a, ptr, hinfo, mx és soa rekordok lekérdezése)
- *HTTP* (mind szerver mind kliens oldal támogatása)
- *SUN RPC* (portmapper, mount, rstat, etherstat, pcnfs szolgáltatások)
- *NTP* (hálózati időszinkronizálási protokoll)
- *UDP* (UDP datagrammok küldése)

A *Tkined* **[New_97]** a Tnm kiterjesztésre épülő Tcl/Tk nyelven írt grafikus hálózatmenedzsment alkalmazás. Mint sok más társa, a Tkined is egy hálózati térkép orientált rendszer. Erre a térképre lehet felvenni a menedzselni kívánt hálózati objektumokat (ill. az *IP Discovery* segítségével ezeket az eszközöket automatikusan is felderíthetjük egy adott címtartományban).

A rendszer magja tulajdonképpen csak a grafikus felület, a térkép és az objektumok kezelését valósítja meg. A hálózatmenedzsmenttel kapcsolatos alapfunkciók is külső modulokba kerültek.

A Tnm könyvtár fejlesztése során **[Sch_95]** beépítettek egy olyan parancsot ill. parancscsaládot *(ined)*, mely hatékonyan támogatja a moduláris alkalmazásfejlesztést és a könnyű bővíthetőséget. A modularitás itt egyrészt jelenti a forrásmodulok felszabdalását, másrészt az egyes modulok külön felhasználói folyamatba (processzbe) szervezését. A folyamatok egy belső protokollon keresztül kommunikálnak a központi alkalmazással (akárcsak az X Window System esetén az alkalmazások a szerverrel). A belső protokollt elrejti az ined parancsfelület (ahogy ezt az Xlib is teszi az X rendszer esetén). A modulok önálló interpreterrel (scotty) futnak, bármit megtehetnek, amit egy szokványos alkalmazás, mellette azonban hozzáférhetnek a rendszermag által nyújtott szolgáltatásokhoz.

A Tnm egy rendkívül jól használható eszköz, komolyabb hibákat, problémákat nem tapasztaltam. Mindenképpen lehet rá alapozni komolyabb munkákat, ahogy ezt sokan meg is teszik.

A Tkined egy érdekes és látványos kezdeményezés, de jó néhány problémával küzd. Nagyon szép koncepció, ahogy a külső modulok és azok integrálása megvalósul, mivel így könnyen továbbfejleszthető az eszköz, másrészt egy-egy hibás modul nem okozza az egész alkalmazás összeomlását. Nem előnyös azonban, hogy minden elindított modul közvetlenül a térkép objektumain dolgozik, mivel nem várt egymásra hatások jelentkeznek. Ehhez kapcsolódó problémája az eszköznek, hogy nem lehet tudni, mikor dolgozik egy modul, mivel erről nincs semmiféle visszajelzés. Így könnyen előfordul, hogy egy megkezdett munkára (pl. IP Discovery) az óvatlan felhasználó „ráindít" egy másik feladatot, ami többnyire hibás működéshez, és az alkalmazás összeomlásához vezet. A Tkined-hez megírt modulok azonban mindenképpen figyelemre méltóak.

A.1.3. OpenLDAP

Az LDAP protokoll első implementációja a Michigan-i Egyetemen kifejlesztett *sldapd (Standalone LDAP daemon)* program volt **[Mich_96]** . Ennek fejlesztése és a protokoll szabványosítása párhuzamosan folyt. Később a protokoll és a program megalkotói a Netscape cégnél folytatták a munkát, és – bár a Netscape Directory Server piacvezető termék lett – a U-Mich LDAP megrekedt a 3.3 verziónál. Hosszú stagnálás után az openldap projekt *(www.openldap.org)* keretében újra önkéntesek foglalkoznak fejlesztésével. Forráskódja szabadon hozzáférhető és UNIX rendszerek között nagyon hordozható. A régóta várt új verzió remélhetőleg támogatni fogja az LDAPv3 protokollt **[RFC2251]** (ami például a kommunikáció titkosítása miatt fontos) és a Windows platformokat.

Az OpenLDAP csomag tartalmaz egy címtár szervert, parancssori klienseket, és egy programozói felületet. Erre a felületre épül az az apró Tcl kiterjesztés is, mely az LDAP szerverek elérését biztosítja Tcl programokból.

A.1.4. MySQL

A felhasznált elemek közül egyedül ennek az eszköznek nem teljesen ingyenes a használata, bár ez csak az üzleti célú felhasználásra vonatkozik. Választásom azért esett mégis erre a termékre, mert egyrészt rendkívül gyors (ami „riasztásviharok" esetén nagyon fontos), könnyen telepíthető, és nagyon sok platformon (beleértve a Windows alapúakat) fut. Gyorsaságának ára, hogy rendkívül korlátozott tranzakció kezelési képességei vannak, amire azonban nem volt égető szükség a projekt során. **[Axm_99]**

A csomag tartalmazza az adatbázis szervert, parancssori klienseket (többek között egy SQL monitor prompt-ot), és – a tcl-sql kiterjesztéshez is szükséges – programozói felületet

A.2. Az EC/M konfigurációs paraméterei

A program konfigurációs beállításairól az ecm.conf file gondoskodik. Megjegyzéseket a sor elején elhelyezett kettős kereszt segítségével tehetünk a file-ba. Az egyes paraméterek jelentését a 7. táblázat tartalmazza.

Könyvtár	Alapétrék	Magyarázat
ecmdir	`/opt/ecm`	*A rendszer gyökér könyvtára*
websrvport	*1974*	*A WEB kiszolgáló TCP portja*
webdatadir	*web/data*	*A WEB kiszolgáló statikus tartalma*
webpidfile	*web/log/websrv.pid*	*A WEB kiszolgáló folyamat azonosítóját tartalmazó file.*
weblogfile	*web/log/websrv.pid*	*A WEB kiszolgáló napló file-ja.*
sqlhost	*localhost*	*Az RDBMS szerver címe*
sqldbname	*ecm*	*Az EC/M által használt adatbázis neve*
sqluser	*ecm*	*Az adatbázishoz történő hozzáférés felhasználói azonosítója*
sqlpasswd		*Az adatbázishoz történő hozzáférés jelszava*
ldaphost	*localhost*	*Az LDAP kiszolgáló címe*
ldapport	*1984*	*Az LDAP kiszolgáló TCP portja*
ldapdn	*o=ECM*	*Az LDAP fa gyökerének DN-je*
ldapbinddn	*cn=Manager, o=ECM*	*Az LDAP adminisztrátor DN-je*
ldappasswd		*Az LDAP adminisztrátor jelszava*
pingintervall	*600*	*Az ICMP-vel megfigyelt eszközök lekérdezési időköze (másodperc)*
pingtimeout	*5*	*Az ICMP-vel megfigyelt eszközök timeout értéke (másodperc)*
pingmonitorpidfile	*monitors/log/pingmonitor.pid*	*A PingMonitor folyamat azonosítóját tartalmazó file*
dnsintervall	*600*	*A DNS szolgáltatás ellenőrzésének lekérdezési időköze (másodperc)*
dnstimeout	*5*	*A DNS szolgáltatáshoz rendelt timeout érték (másodperc)*
dnsmonitorpidfile	*monitors/log/dnsmonitor.pid*	*A DNSMonitor folyamat azonosítóját tartalmazó file*
webintervall	*600*	*A WEB szolgáltatás ellenőrzésének lekérdezési időköze (másodperc)*
webtimeout	*5*	*A WEB szolgáltatáshoz rendelt timeout érték (másodperc)*
webmonitorpidfile	*monitors/log/webmonitor.pid*	*A WebMonitor folyamat azonosítóját tartalmazó file*
trapmonitorpidfile	*monitors/log/trapmonit*	*A TrapMonitor folyamat azonosítóját*

	or.pid	*tartalmazó file*
routerintervall	*600*	*A routerek interfészei ellenőrzésének lekérdezési időköze (másodperc)*
routertimeout	*5*	*A routerek lekérdezéséhez rendelt timeout érték (másodperc)*
routermonitorpidfile	*monitors/log/routermonitor.pid*	*A RouterMonitor folyamat azonosítóját tartalmazó file*
ecdintervall	*600*	*A korrelációs tevékenység frekvenciája (másodperc)*
ecdmaxage	*7*	*Az elévült riasztásbejegyzések kora (nap)*
ecdpidfile	*ecd/log/ecd.pid*	*A korrelációs folyamat azonosítóját tartalmazó file*
ecddelta	*3600*	*Az összetartozó azonos események között eltelhető maximális idő (másodperc)*

7. táblázat Konfigurációs paraméterek

A.3. A CDROM melléklet tartalma

A mellékelt CD tartalmazza az elkészített alkalmazást, a felhasznált eszközök forrását, az elektronikusan is elérhető hivatkozott irodalmat, valamint a dolgozatban szereplő OpenView NNM szoftver próba változatát.

Az alkalmazás telepítése nem egyszerű feladat, de Linux vagy Solaris platformokon könnyebben megoldható. Ehhez célszerű létrehozni egy *ecm* nevű felhasználót, melynek HOME könyvtára az */opt/ecm* könyvtár. A megfelelő bináris csomagot ide kell kibontani. A program használata előtt az *sh.rc* file használata *(„source /opt/ecm/sh.rc")* előnyös, mely beállítja a futtatáshoz szükséges környezeti változókat. Természetesen a működéshez szükséges az LDAP objektumtár megfelelő feltöltése (részben a discover programmal, részben kézzel az *ldapadd*, *ldapmodify* parancsokkal, vagy egy tetszőleges LDAP böngészővel.)

A CD könyvtárait mutatja a 8. táblázat.

Könyvtár	Tartalom
ecm/src	*Az alkalmazás általam készített vagy módosított (websrv) moduljai*
ecm/packages	*Az alkalmazásban felhasznált külső eszközök forrásai*
ecm/binaries	*A teljes alkalmazás bináris formában Linux és Solaris platformokra tar.gz formátumban*
docs/	*A hivatkozott irodalom elektronikusan elérhető része*
ov/	*Az OpenView NNM szoftver próba (eval) verziója*
diploma/	*A diplomaterv elektronikus alakban (PDF)*

8. táblázat A CDROM melléklet könyvtárai

IRODALOMJEGYZÉK

[Out_94] Outerhout, John *Tcl and the Tk Toolkit*
Addison-Wesley, 1994

[Zelt_98] Zeltserman, Dave - Pioplo *Building Network Management Tools with Tcl/Tk*
Prentice Hall, 1998

[Sta_99] Stallings, William *SNMP, SNMPv2, SNMPv3, RMON1and 2 3rd Ed.*
Addison-Wesley, 1999

[Har_98] Harrison, Mark*Tcl/Tk Tools*
O'Relly & Associaties, 1998

[Wel_99] Welch, Brent *Practical Programming in Tcl and Tk 2nd Ed.*
Prentice Hall, 1999

[Ball_99] Ball, Steve *Web Tcl Complete*
McGraw-Hill, 1999

[Tre_95] Treplán Kornél *Lokális hálózatok menedzselése*
Panem, 1995

[She_94] Sheers, R. Kenneth *HP OpenView Event Correlation Services*
HP Journal, 1994 Oct.

[Sch_95] Schönwälder, J. – Langendörfer, H. *Tcl Extension for Network Management Applications*
Proc. 3rd Tcl/Tk Workshop, Toronto, July 1995

[Sch_93] Schönwälder, J. – Langendörfer, H. *How To Keep Track of Your Network Configuration*
Proc. 7th Conf. on Large Installation System Administration, Monterey, Nov 1993

[New_97] Newnham, Mark *Getting Started with Tkined*
Jan. 1997

[Wil_99] Wilcox, Mark *Implementing LDAP*
Wrox Press, 1999

[HP_00] HP, *Managing Your Network with HP OpenView Network Node Manager*
Hewlett-Packard Comp., Jan. 2000.

[RFC1155] Rose, M. – McClogherie, K. *Structure and Identification of Management Information for TCP/IP*
RFC1155, May. 1990

[RFC1157] Case, J. – Fedor, M.- Davin, J. *A Simple Network Management Protocol (SNMP)*
RFC1157, May. 1990

[RFC1212] Rose, M. – McClogherie, K. *Concise MIB Definitions*
RFC1212, Mar. 1991

[RFC1213] Rose, M. – McClogherie, K. *Management Information Base for Network Management of TCP/IP based internets: MIB-II*
RFC1213, Mar. 1991

[RFC1215] Rose, M.. *A Convention for Defining Traps for use with the SNMP*
RFC1215, Mar. 1991

[RFC1777] Yeong, W. – Howes, T. – Kille, S. *Lightweight Directory Access Protocol*
RFC1777, Mar. 1995

[RFC1778] Yeong, W. – Howes, T. – Kille, S. *The String Representation of Standard Attribute Syntaxes*
RFC1778, Mar. 1995

[RFC1779] Kille, S. *The String Representation of Distinguished Names*
RFC1779, Mar. 1995

[RFC2251] Wahl, M. – Howes, T. – Kille, S. *Lightweight Directory Access Protocol v3*
RFC2251, Dec. 1997

[RFC2253] Wahl, M. – Howes, T. – Kille, S. *UTF-8 String Representation of Distinguished Names*
RFC2253, Dec. 1997

[RFC2254] Howes, T. *The String Representation of LDAP Search Filters*
RFC2254, Dec. 1997

[RFC2255] Howes, T. *The LDAP URL Format*
RFC2255, Dec. 1997

[Kem_79] Kemény, J. *The President's commission on the accident at Three Mile Island*
Government Printing Office, 1979

[Cis_99] Cisco Systems Inc. *Cisco Info Center 1.2 Release User Guide*
1999.

[Axm_99] Axmark, D. – Widenius, M. – DuBois, P. *MySQL Reference Manual*
T.c.X. AB 1999.

[Mich_96] *The SLAPD and SLURPD Administrator's Guide*
University of Michigan 1996.

RÖVIDÍTÉSEK

ASN.1.	Abstract Syntax Notation One
BER	Basic Encoding Rules
CMIP	Common Management Information Protocol
HTTP	Hyper Text Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
MIB	Management Information Base
OSI	Open Systems Interconnection
OID	Object Identifier
PDU	Protocol Data Unit
PKI	Public Key Infrastructure
RFC	Request for Comment
RPC	Remote Procedure Call
RMON	Remote Network Monitoring
SMI	Structure of Management Information
SNMP	Simple Network Management Protocol
SSL	Secure Socket Layer
TCP	Transmission Control Protocol
TFTP	Trivial File Transfer Protocol
UDP	User Datagram Protocol
WAN	Wide Area Network
WBMA	Web Based Management Application