Informatikai technológiák laboratórium 2

Szolgáltatási szint menedzsment

Mérési segédlet

Készítette: Kocsis Imre

ikocsis@mit.bme.hu

2009. november 17.

Verzió: 3.1

Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

1 Bevezető

A mérés célja, hogy a hallgatók megismerkedjenek az informatikai infrastruktúrák segítségével megvalósított szolgáltatások szolgáltatási szint menedzsmentjének alapelemeivel. Ennek érdekében a mérés során megismerkedünk egy központosított esemény-feldolgozást támogató eszközzel, azt összekötjük egy egyszerű elosztott példainfrastruktúrát és annak szolgáltatásait monitorozó instrumentációval, majd az esemény-feldolgozást úgy konfiguráljuk, hogy az támogassa a szolgáltatási szintek követését és néhány manuálisan injektált hiba diagnosztikáját.

A mérési segédlet röviden tárgyalja a szolgáltatási szinteket, mint általános fogalmat, azok menedzsmentjének tipikus feladatait és áttekintést nyújt a mérésen használt eszközről; nem célja azonban a konkrét mérési feladatok és a mérőkörnyezet ismertetése.

Az ipari gyakorlatban a mérésen a hallgatók által használt eseményfeldolgozó-keretrendszert nem közvetlenül szokás szolgáltatási-szint menedzsmentre használni, hanem egy célspecifikus termék, a Tivoli Business Service Manager [5] (TBSM) azon komponenseként, mely a beérkező riasztásokat szolgáltatás-komponens állapotokra konvertálja. A TBSM ezen állapotinformációk segítségével a szolgáltatásokat mint szolgáltatás-komponensek függőségi fáját grafikusan ábrázolja, azt az állapotok segítségével színezi és SLA-kiértékeléseket végez. Bár a mérés időkorlátaira való tekintettel nem áll módunkban a TBSM-mel mérési feladatokat végrehajtani, a mérés lezárásaként annak működését a mérésvezetők röviden demonstrálni fogják.

A mérésre való felkészülés részeként – a mérési segédlet feldolgozása után – kérjük, hogy olvassák el a Netcool/OMNIbus hivatalos dokumentációjából az eseménygyűjtő szondák működését konfiguráló egyszerű szkriptnyelv szintaxis-referenciáját is. [8] Bár ezt a beugró során nyilvánvalóan nem kérjük számon, áttekintési szintű ismerete szükséges a mérési feladatok ütemes megoldásához.

2 Szolgáltatási szintek az informatikában: nemfunkcionális attribútumok metrikái és mérésük

Az informatikai infrastruktúrával megvalósított szolgáltatásoknak közös jellemzője, hogy legtöbbször találhatóak olyan jellemző mérőszámok, melyek a felhasználó számára a szolgáltatás minőségét (*Quality of Service, QoS*) mérik. Az IT szolgáltatások esetén ezeknek a mérőszámoknak a definícióját, a rájuk vonatkozó megkötéseket és be nem tartásuk esetére vonatkozó megállapodásokat szokás szolgáltatási szint szerződésekben (*Service Level Agreement, rövidítve SLA*) rögzíteni. Az SLA-k tipikusan jogi erővel rendelkező dokumentumok; az SLA be nem tartása általában kártérítésre kötelezi a szolgáltatót.

Egy rendszer megfelelőségének funkcionális és nemfunkcionális (más szóhasználattal extrafunkcionális) aspektusait szokás megkülönböztetni. IT rendszerek esetén a nemfunkcionális aspektuson belül szokásosan a szolgáltatásbiztonság (*dependability*), adatbiztonság (*security*) és teljesítmény (*performance*) szempontjait különböztetjük meg. Az adatbiztonság és a szolgáltatásbiztonság jellemzésére azokhoz tipikusan a következő attribútumokat rendeljük:

- szolgáltatásbiztonság:
    - rendelkezésre állás (*availability*): készenlét (helyes) szolgáltatás nyújtására
    - megbízhatóság (*reliability*): helyes szolgáltatás nyújtásának folytonossága
    - biztonságosság (*safety*): a felhasználók és környezetük szempontjából a katasztrofális hatások hiánya (főleg biztonságkritikus, pl. vasúti irányító rendszereknél fontos)
    - integritás (*integrity*): a rendszer helytelen módosulásainak hiánya
    - karbantarthatóság (*maintainability*): módosíthatóság és javíthatóság
- adatbiztonság:
    - rendelkezésre állás
    - integritás
    - bizalmasság (*confidentiality*): jogosulatlan információszerzés hiánya

Az egyes attribútumok pontosabb definícióját lásd a hivatkozott irodalomban [1].

Ezek az attribútumok azonban nem mérőszámok (az ipari szóhasználatban metrikák); sok esetben komoly mérnöki munkát jelent azon mérhető, vagy mért értékekből származtatható metrikák definiálása, melyek a fenti attribútumokat egy adott rendszeren, az adott alkalmazási esetben jól reprezentálják. Bizonyos mérnöki területeken, mint például az alacsony szintű telekommunikációs szolgáltatások esetén léteznek szabványosnak tekinthető, illetve szabványosított metrika készletek; általános esetben azonban az SLA-ban a fenti attribútumok, illetve a teljesítmény kifejezésére alkalmazandó metrikák körének megállapítása mind üzleti, mind mérnöki jellegű, eseti analízist igényel.

Az absztrakt attribútumok és a konkrét metrikák kapcsolatának megvilágítására példaként tekintsünk egy statikus webhoszting szolgáltatást. Milyen nemfunkcionális igényei lehetnek a vevőnek és mik ennek a mérőszámai?

Rendelkezésre állás: a weboldal legyen mindig elérhető, amikor meg akarják nézni (tehát például az éjszaka közepén nem olyan fontos…). Mértéke lehet egy „availability on demand” mérőszám (pl. egy órás csúszóablakos számítással a fogadott kapcsolódási kísérletek száma osztva az összes kapcsolódási kísérletek számával). Az „on demand” jelleg azonban csak nehézkesen mérhető (honnan tudnánk, hogy mikor melyik látogatási kísérlet nem sikerült, ha nincs komolyabb szintű hozzáférésünk a szolgáltató infrastruktúrájához?). Egyszerűbb megoldás a rendelkezésre állás tényét a szolgáltató infrastruktúrájához képest „kívülről”, periodikus lekérdezéssel ellenőrizve időintervallumra vett rendelkezésre állási valószínűséget számolni és különböző időintervallumokra különböző korlátokat megszabni. Mindezek függvényében a rendelkezésre állást definiálhatjuk például a következőképpen:

$$A = \frac{\sum_{i=-1}^{-n} p_i}{n},$$

ahol n a jelenlegi, (0-nak tekintett) időpillanathoz képest a folytatólagos múltban figyelembe venni kívánt periodikus mérések száma, $p_i \in \{0,1\}$ pedig az i-edik mérés során tapasztalt megfigyelés (a szolgáltatás elérhető-e vagy sem). A metrika egy számított értékének értelmezési intervalluma az a $[t_0 - T, t_0]$ intervallum, ahol t_0 a fenti számításban „jelenként" értelmezett időpillanat és

$$T = n\frac{1}{f},$$

ahol f a mérési frekvencia.

A fenti metrika-definícióval kapcsolatban a következőket érdemes észrevenni.

1) A t_0 időpillanatban mért értéket nem vesszük bele a $[t_0 - T, t_0]$ időintervallumhoz tartozó metrika-érték számításokhoz felhasznált mérési eredmények halmazába. Ez azt jelenti, hogy a $[t_{-i-1}, t_{-i}]$ intervallumra a rendelkezésre állást megfeleltetjük a t_{-i-1} időpillanat mérési értékével. „Fordítva" is eljárhatnánk persze, vagy vehetnénk az intervallum két korlátjánál mért érték átlagát; elégségesen gyakori mintavételezésnél azonban mondhatjuk azt, hogy lényegtelen, hogy melyik megközelítést használjuk. Vegyük észre, hogy a jelek periodikus mintavételezésére vonatkozó összefüggések itt is érvényesek, azaz a mintavételezés frekvenciája alapvetően befolyásolja a mérés pontosságát. Az is igaz azonban, hogy a mintavételezés erőforrásokat használ; a frekvencia megállapításánál tehát szükségszerűen kompromisszumokat kell kötnünk.
2) Valójában már azzal pontatlanságok lehetőségét vezettük be, hogy egy külső ágens általi mintavételezés mellett döntöttünk. Egy web szolgáltatás ugyanis túlterheltség (gondoljunk a „Slashdot effect"-re, vagy egy DDOS támadásra) esetén a kéréseknek egy részét fogja visszautasítani illetve a kapcsolódási kísérletek egy része lesz sikertelen. Nem csak hogy nem tudjuk, hogy a mintavételezésünk az egyes mintavételi időpontokban melyik kategóriába esik, de (mint arra fentebb utaltunk) azt sem tudjuk, hogy a kérések mekkora része sikeres és mekkora nem.

Megbízhatóság: amikor egy kapcsolódási kísérlet sikeres, akkor a helyes tartalom, tehát „a mi oldalaink" kerüljenek kiszolgálásra; a megfontolások ugyanazok mint fentebb. (Azzal a kiegészítéssel, hogy esetünkben az adatbiztonság integritás attribútuma és a szolgáltatásbiztonság megbízhatósági attribútuma láthatóan összeérnek.)

Teljesítmény: üzleti szempontból legjobb lenne azt mérni, hogy a böngésző kliensek milyen gyorsan végeznek az oldal betöltésével (ugye emlékszünk még a klasszikus „7 másodperces" szabályra, mely a szélessávú internetelérések elterjedésével mára inkább 3-4 másodpercre csökkent?), ehhez azonban olyan elemeken kellene mérni, melyekhez jellemzően nincs instrumentációnk (a böngésző a kliens gépen, de legalábbis a szervertől a kliensig terjedő teljes hálózat). Ami szolgáltatói oldalon realisztikusan pl. szerver logokból kideríthető: a HTTP kiszolgáló által a kérés kiszolgálásának megkezdése és befejezése között eltelt idő átlaga valamilyen intervallumra nézve. További instrumentáció segítségével (pl. fejlettebb hálózati kapcsolóelemekbe illetve útvonalválasztókba épített, TCP kapcsolat szintű monitorozás) ez kiterjeszthető úgy, hogy a TCP kapcsolat felépítési kísérlettől kezdve a kiszolgálás befejezéséig mérjük az eltelt időt. Műszakilag itt is egyszerűbb megoldás azonban, ha külső ágenssel mintavételezünk periodikusan és a felmerülő pontatlansági források tudatában ebből következtetünk az összes kéréskiszolgálás teljesítményjellemzőire. Erre a feladatra rengeteg céleszköz elérhető; egyszerű, nyílt forráskódú megoldásoktól (mint pl. az openwebload [2]) a felhasználói viselkedést session szinten emuláló nagyvállalati termékekig (pl. IBM Tivoli Composite Application Manager for Transactions [3]).

Figyeljük meg a teljesítmény esetén intuitívan felvett metrika kompozit jellegét; a szolgáltatás nyújtásában részt vevő elemek (pl. hálózati hozzáférés, szerver erőforrások, operációs rendszer, kiszolgáló alkalmazás) szokványos teljesítmény mérőszámai – mint például a hálózati réteg esetén az IP szintű késleltetés és áteresztőképesség – külön-külön nyilván hatással vannak rá, de a szolgáltatási szintű metrika származtatása belőlük nem triviális.

3 Szolgáltatási szintek menedzsmentje

Az eddigiekben áttekintettük, hogy egy SLA-ban mely nemfunkcionális rendszerattribútumokat leíró metrikákra szokás korlátokban megállapodni és példákat adtunk az üzleti igényekből következő, mérhető szolgáltatási szintű metrikákra. Milyen felügyeleti feladatok kapcsolódhatnak magukhoz az SLA-khoz, illetve olyan rendszerekhez, melyeknek SLA korlátoknak megfelelő szolgáltatást kell nyújtaniuk?

Az SLA-k, mint szerződések életciklus-menedzsmentje. Az életciklus-menedzsment az SLA-khoz kapcsolódó adminisztratív feladatok megvalósítását jelenti. Léteznek szoftvereszközök, melyek az SLA-k megkötését, karbantartását, betartásuk követését és megszüntetését támogatják.

SLA betartásának követése. Az SLA aktuális állapotának követése – pl. egy adott hónapban egy vállalat által előfizetett bérelt vonalon mennyi „maradt még" az SLA által egy hónapra megengedett szolgáltatás-kiesésből – mind az ügyfélnek, mind a szolgáltatónak nyilvánvaló érdeke. Az ügyfél oldalán érdemes olyan egyszerű, a *szolgáltatótól független* mérőinfrastruktúrát kialakítani, mely megfelelően részletes visszamenőleges adatgyűjtés mellett (ez az utólagos bizonyításhoz lehet szükséges) rögzíti az SLA-ban definiált metrikák értékeinek idősorait. Az SLA állapotkövetést szolgáltatói oldalon tanácsos komolyabb, adattárház jellegű historikus adatgyűjtéssel kiegészíteni, mely lehetővé teheti az SLA sértések előrejelzését. (Az IBM Tivoli termékpalettában erre a célra szolgál az IBM Tivoli SLA Advisor [4].)

Informatikai folyamatok SLA állapot vezérelt végrehajtása. Egy jól felügyelt informatikai infrastruktúrán általában rendelkezésre állnak automatizált IT menedzsment folyamatok (mint pl. egy szervert távolról a „bare metal" szintről újratelepítő és konfiguráló mechanizmus), melyek legtöbb esetben emberek által végrehajtott felügyeleti folyamatok lépései (gondoljunk csak az ITIL-re, ami legnagyobbrészt ilyen „best practice" folyamatok gyűjteménye). Amennyiben a felügyelt infrastruktúra által megvalósított szolgáltatásoknak SLA korlátokat kell betartaniuk, úgy az IT felügyeleti folyamatokat is úgy kell kialakítani, hogy megfelelő mértékben biztosított legyen az SLA sértések elkerülése. Egy példa: egy SLA-val ellátott szolgáltatást futtató szerver hibájának manuális, szakértői diagnosztizálása és javítása a hónap első felében mindaddig megengedett, amíg az adott hónapra megengedett kiesésből még hátravan legalább öt óra; ha ezen időkorlát alá jutunk, úgy a diagnosztizálás és javítás állapotától függetlenül automatikusan be kell állítani egy új szervert a kiesett helyére, ismerten jól működő szoftverkonfigurációval, mivel egy tartalékszerver vagy egy bérelt szerver ezen célra való átmeneti beállítása olcsóbb, mint az SLA megsértése. Érdemes megjegyezni, hogy az előző pontokban műszaki jellegű felügyeleti feladatként tárgyalt életciklus-menedzsment és SLA megfelelőség követés folyamat-vetületei egy jól megtervezet folyamat-rendszernek részei kell, hogy legyenek (lásd ITIL).

„Kiváltó okok keresésének" (root cause analysis) támogatása. Az SLA megfelelőséget az SLA állapot-követés és a megfelelő folyamatok önmagukban még nem biztosítják. Szükséges az is, hogy a szolgáltatás referencia viselkedéstől való eltérésének okait be tudjuk határolni, mégpedig

a) elég gyorsan ahhoz, hogy az SLA sérüléséig hátralévő időben még meg tudjuk szüntetni az okokat;

b) megbízhatóan, azaz a kiváltó okok keresése ne, vagy csak ritkán jusson rossz eredményekre; és

c) elegendő felbontással ahhoz, hogy rendelkezésre álljanak a költségek szempontjából is végrehajtható akciók (példa: ha egy diagnosztikai megoldásban a diagnosztika felbontása egy fizikai kiszolgáló bináris hibamóddal, úgy további vizsgálatok nélkül nem tudjuk megkülönböztetni egy szoftverkomponens újraindítással javítható hibáját például a merevlemez permanens hibájától – a lehetséges javító akciók ennél a felbontásnál az újraindítás (ami kevert diagnosztikai/javító akció, hiszen sikertelensége diagnosztikai információt is hordoz) illetve a szerver cseréje).

Ezek a megfontolások igazak mind a klasszikus szolgáltatásbiztonsági diagnosztikára (ahol *hibaokok* vezetnek a rendszerkomponensek *hibás állapotaihoz*, melyek a komponens által megvalósított szolgáltatások referenciától eltérő viselkedésében manifesztálódhatnak *hibahatásként*), mind a teljesítményproblémák okainak feltárására.

4 IBM Tivoli Netcool OMNIbus

Az IBM Tivoli Netcool/OMNIbus egy központosított, valósidejű[1] eseménygyűjtő és -feldolgozó szolgáltatás- és rendszerfelügyeleti keretrendszer. Központi eleme az ún. ObjectServer, mely eseményjelzéseket gyűjt a felügyelt szolgáltatásokról és infrastrukturális elemekről. A beérkezett események feldolgozása (pl. korreláció, duplikált jelzések szűrése, riasztások kiadása/megszüntetése) széles keretek között konfigurálható, minek köszönhetően egyfajta „intelligens riasztási központként" a legkülönbözőbb felügyeleti feladatok ellátására képessé tehető, a hibás rendszerkomponensek jelzésétől kezdve a (fél)automatizált diagnosztikán keresztül a szolgáltatási szint megfelelőségek követéséig. Az egyéb Tivoli termékekkel való integráció mellett eseményforrásként számos saját ágens-típust is képes használni.

Ezen segédletben az OMNIbus-szal kapcsolatban a leglényegesebb tudnivalókra szorítkozunk; az érdeklődő Olvasó figyelmébe ajánljuk a [6]-os hivatkozást. A fejezet fő célja, hogy bemutassa az OMNIbus számunkra lényeges komponenseit és a kapcsolódó adminisztrációs/felhasználói eszközöket. Nem foglalkozunk például az OMNIbus néhány, egyébként fontos komponensével és a különböző konfigurációs nyelvek tárgyalását is részben elhanyagoljuk.

4.1 ObjectServer

Az ObjectServer egy memóriában megvalósított adatbázis-kiszolgáló (in-memory database), melyből Windows platformon alapértelmezett telepítés esetén egy példány jön létre (a belső elnevezés: „`NCOMS`", a megfelelő Windows szolgáltatás neve: `NCOObjectServer`). Az adatbázis-kiszolgálón több alapértelmezett adatbázis is települ; ezek közül a számunkra legfontosabb az `alerts` adatbázis, annak is az `alerts.status` táblája. Ennek minden egyes sora egy eseményt (illetve egy „riasztást") reprezentál. A fontosabb mezőket az 1. táblázat foglalja össze (ezeket részben az ObjectServer tartja karban, mint azt később látni fogjuk; a teljes sémát lásd: [7]).

Az adatbázis-séma szintjén az `Identifier` oszlop érdemel figyelmet. Az `Identifier` oszlopra a következő oszlopattribútumok (flagek) `true` értékűek: `Primary Key`, `No Default`, `No Modify`, azaz az oszlop elsődleges kulcs, egy sor első beszúrásánál mindenképpen kell az `Identifier`–nek értéket adni és az érték később nem módosítható. A többi oszlop esetén nincs `true` értékű speciális flag (kivéve a `RowID` és `RowSerial` oszlopokat, ám azokat az adatbázismotor kezeli).

1. táblázat: az ObjectServer alerts.status táblájának főbb oszlopai

Oszlopnév	Adattípus	Leírás
`Identifier`	`varchar(255)`	Egyértelmű eseményazonosító, mely pl. a deduplikációt vezérli; lásd lentebb
`Node`	`varchar(64)`	A riasztás forrásaként értelmezhető felügyelt entitás. IP hosztok esetén a hoszt (DNS) neve; amennyiben az nem állapítható meg, úgy az IP cím.
`Manager`	`varchar(64)`	Az eseményt begyűjtő probe (lásd később) neve; használható a probe-ot futtató hoszt jelzésére is
`AlertKey`	`varchar(255)`	Az adott node-on belül annak a felügyelt entitásnak a neve, amire a

[1] A szó „puha" értelmében; azaz az esemény-feldolgozás a historikus módszereken alapuló, illetve a legfeljebb kb. perces felbontással operáló megoldásokhoz képest kvázi valósidejű, ám a „hard realtime" fogalom alatt általában értett időkorlát-betartást nem biztosít.

		riasztás vonatkozik
Severity	integer	Súlyosság érték; meghatározza a riasztás színét az eseménylistában (lásd később). Az értelmezett konstansok: 0: „clear” („törölhető” üzenet) 1: nem meghatározott 2: figyelmeztetés 3: kisebb probléma 4: súlyos probléma 5: kritikus
Summary	varchar(255)	A riasztás okának szöveges összefoglalója.
StateChange	time	Automatikusan karbantartott időbélyeg (lásd a triggereket később); az adott sor legutolsó beszúrásának vagy frissítésének időpillanata (akármely forrásra)
FirstOccurence	time	A riasztás első beszúrása és a UNIX epoch között eltelt idő, másodpercekben
LastOccurence	time	Az utolsó időpont amikor a riasztás frissítve lett az őt szolgáltató szondán (lásd később)
InternalLast	time	Az utolsó időpont amikor a riasztás frissítve lett az ObjectServer-en
Tally	integer	Az ezen a riasztáson bármely forrásból elvégzett (újra)beszúrások és frissítések száma (lásd deduplikáció)
Acknowledged	integer	0/1 flag; azt reprezentálja, hogy az eseményt egy operátor már tudomásul vette-e vagy sem
ExpireTime	integer	Az esemény törlődjön automatikusan ennyi másodperc után
Type	integer	A riasztás típusa (figyelem: ez nem azonos a súlyossággal!). A fontosabb értelmezett konstansok: 0: nem beállított 1: probléma 2: probléma megoldása

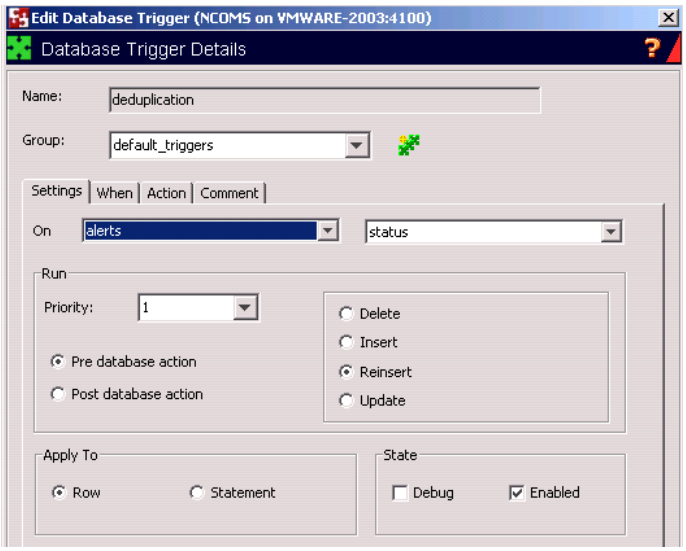
Tehát minden egyes, akármilyen forrásból származó esemény, illetve „riasztás” egy adatbázis-sorrá képződik le; a felügyelt elosztott rendszer eseményeiről alkotott képünk karbantartását így a relációs

adatbázisok szokásos sorszintű atomi műveleteivel kellene megoldanunk (`INSERT/REINSERT`, `UPDATE` és `DELETE`).

Ezek a műveletek azonban önmagukban nem elégségesek több, az esemény-feldolgozásban szokásos feladat végrehajtására, mint például a duplikált események szűrése vagy az elévült események törlése. Az OMNIbus esetén ezek a funkciók is az „adatbázison belül", adatbázis triggerekként[2], nem pedig adatbázis-kliensben kerülnek jellemzően megvalósításra.

Az ObjectServer adatbázismotorja három fajta triggert különböztet meg: adatbázis (`database`), időzített (`temporal`) és jelzés (`signal`).

Adatbázis triggerre nézzük a következő, az alapértelmezett konfigurációval települő példát:



1. ábra: a deduplication trigger beállításai

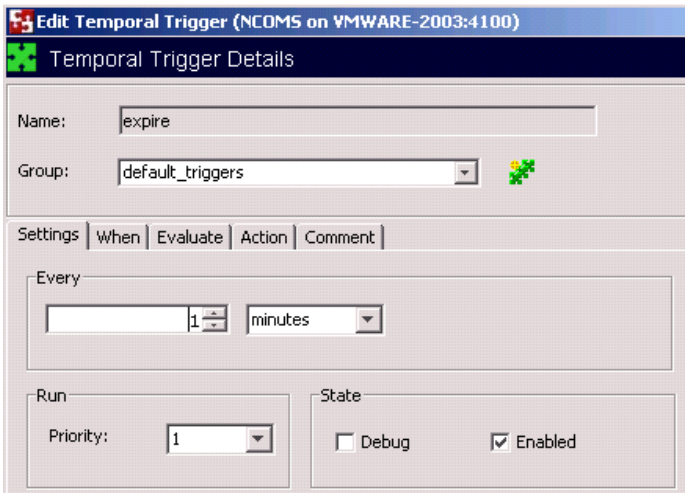
Azaz a 'deduplication' trigger az `alerts.status` táblán értelmezett és annak soraira vonatkozó `REINSERT` operáció esetén tüzelhet. A 'When' fül üres; az 'Action' fül alatt a következő triggerprogram található:

```
begin
        set old.Tally = old.Tally + 1;
        set old.LastOccurrence = new.LastOccurrence;
        set old.StateChange = getdate();
        set old.InternalLast = getdate();
        set old.Summary = new.Summary;
        set old.AlertKey = new.AlertKey;
        if (( old.Severity = 0) and (new.Severity > 0))
        then
                set old.Severity = new.Severity;
        end if;
end
```

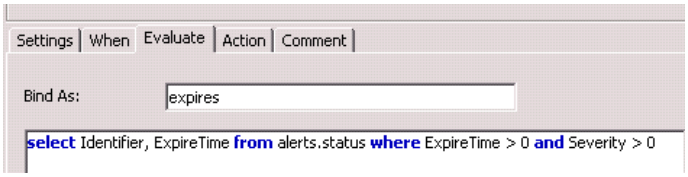
Tehát amennyiben egy már létező elsődleges kulccsal tárolt riasztást próbálunk (újra) beszúrni, úgy azt nem cseréljük le, hanem még a `REINSERT` lefutása előtt „elkapva" az operációt az újonnan beszúrni kívánt sor néhány attribútuma alapján megváltoztatjuk az eredeti sor néhány régi attribútumát (minden más ami módosult volna elveszik).

[2] Lásd pl. http://en.wikipedia.org/wiki/Database_trigger

Időzített triggerre példák az 'expire' és a 'delete_clears' (ezek szintén az alapértelmezett konfigurációval települnek).



2. ábra: expire trigger

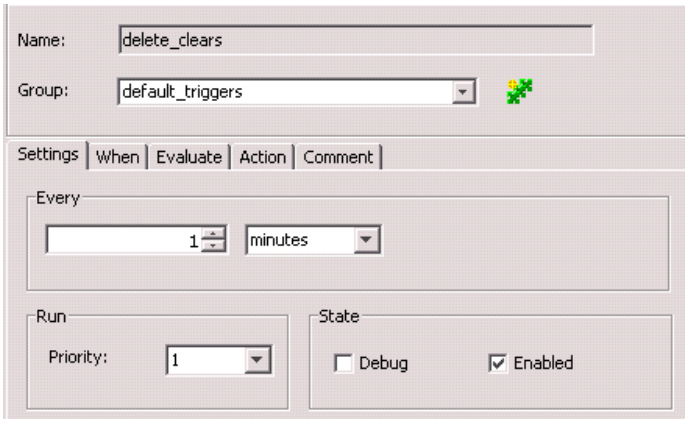


3. ábra: expire trigger

Az expire trigger által végrehajtott akció:

```
for each row expire in expires
begin
        update alerts.status via expire.Identifier set Severity = 0 where LastOccurrence < (getdate() -
        expire.ExpireTime);
end;
```

Az így 0-ra átállított súlyosságú (és a többi „információs") riasztásokat a delete_clears időzített trigger kezeli:



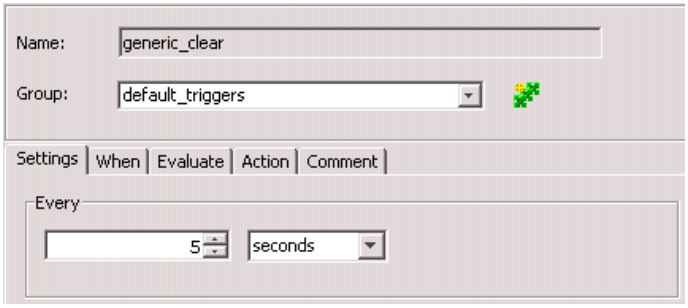
4. ábra: delete_clears trigger

A delete_clears trigger által végrehajtott akció:

```
begin
        delete from alerts.status where Severity = 0 and StateChange < (getdate() - 120);
end
```

Látható, hogy a szabadon konfigurálható adatbázis-technológiának köszönhetően mind az eseménytárolás, mind az események feldolgozása testre szabható és kiegészíthető, például saját esemény-korrelációval. (Valójában az adatbázis triggereken kívül az adatbázismotor lehetőséget nyújt más jellegű automatizmusok használatára is, ám azokkal itt nem foglalkozunk.)

Utolsó példaként tekintsük még át, hogy az alapértelmezett konfiguráció hogyan támogatja az eseménykorrelációt a '`generic_clear`' időzített triggerrel!



5. ábra: generic_clear triger

Az öt másodpercenként végrehajtott akció:

```
begin
        -- Populate a table with Type 1 events corresponding to any uncleared Type 2 events
        for each row problem in alerts.status where
                                problem.Type = 1 and problem.Severity > 0 and
                (problem.Node + problem.AlertKey + problem.AlertGroup + problem.Manager) in
                ( select Node + AlertKey + AlertGroup + Manager from alerts.status where Severity > 0 and
                        Type = 2 )
        begin
                insert into alerts.problem_events values ( problem.Identifier, problem.LastOccurrence,
                                                        problem.AlertKey, problem.AlertGroup,
                                                        problem.Node, problem.Manager, false );
        end;

        -- For each resolution event, mark the corresponding problem_events entry as resolved
        -- and clear the resolution
        for each row resolution in alerts.status where resolution.Type = 2 and resolution.Severity > 0
        begin
                set resolution.Severity = 0;
                update alerts.problem_events set Resolved = true where
                                LastOccurrence < resolution.LastOccurrence and
                                Manager = resolution.Manager and Node = resolution.Node and
                                AlertKey = resolution.AlertKey and AlertGroup = resolution.AlertGroup ;
        end;
```

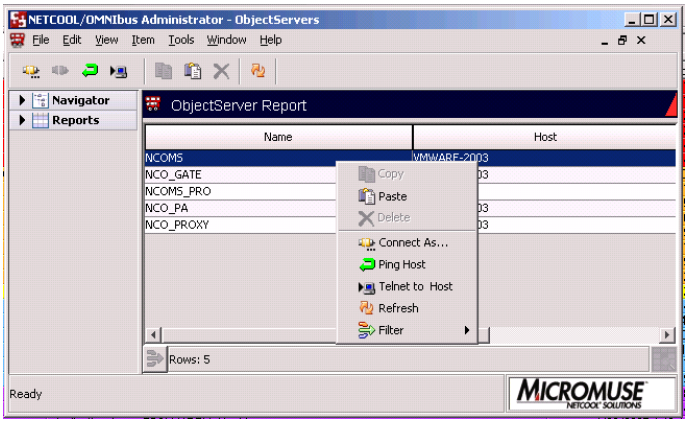
```
        -- Clear the resolved events
        for each row problem in alerts.problem_events where problem.Resolved = true
        begin
                update alerts.status via problem.Identifier set Severity = 0;
        end;

        -- Remove all entries from the problems table
        delete from alerts.problem_events;
end
```

A kódban megvalósított korrelációs logika a következő: egy segédtáblába gyűjtsünk ki minden olyan „Type 1” (azaz probléma) eseményt, mely még nem lett törlésre jelölve (`Severity > 0`) és amelyhez található olyan „Type 2” (azaz probléma-megoldás) esemény, melyet még nem jelöltünk törlésre. A probléma – megoldás párosítás a `Node`, `AlertKey`, `AlertGroup` és `Manager` mezők egyezése alapján történik. Ezután minden „Type 2” eseményt jelöljünk törlésre és jelöljük meg azokat a párosítható problémákat a segédtáblában, melyek legutoljára előbb történtek, mint a probléma-megoldás esemény. Végül minden, a segédtáblában megjelölt eseményt jelöljünk törlésre az `alerts.status` táblában és töröljük a segédtáblát.

4.2 Netcool/OMNIbus Administrator

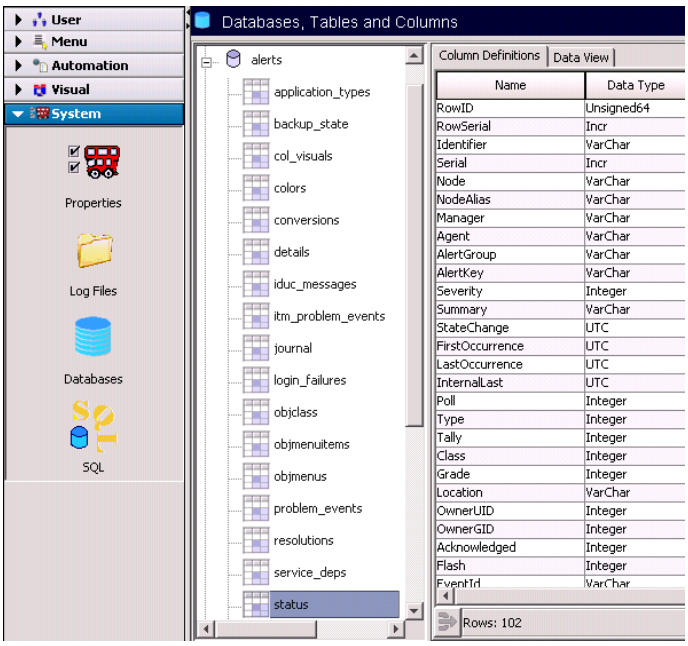
Az Administrator alkalmazás teszi lehetővé az OMNIbus szerver oldali komponenseinek grafikus felületen való kezelését. Alapértelmezett telepítés esetén indítása: **Start -> All Programs -> Netcool Suite -> Administrator**. Az alapértelmezetten megjelenő komponens listában egy ObjectServer felugró menüjében a **„Connect As...”** opcióval csatlakozhatunk az adott ObjectServer-hez.



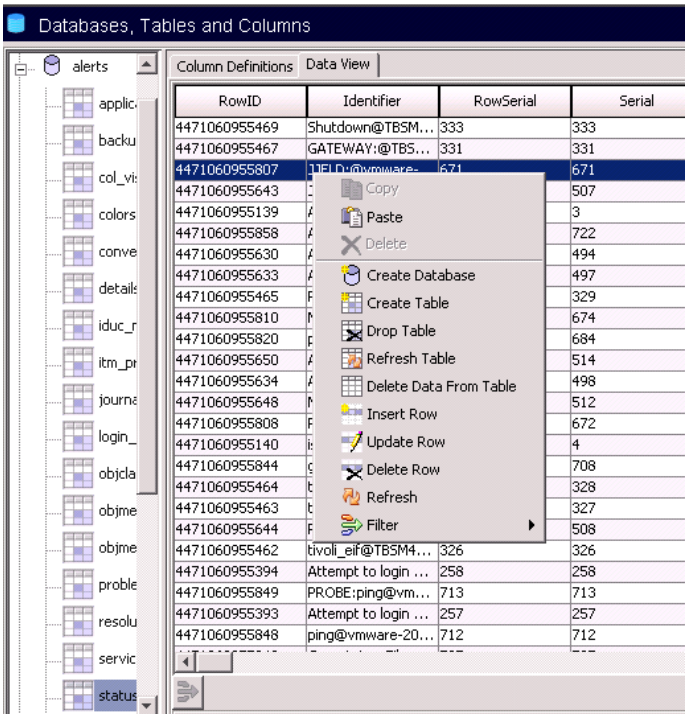
6. ábra: Csatlakozás egy ObjectServerhez

A megjelenő ablakban a `'System'` fül alatt a `'Databases'` pontot választva jutunk az adatbázisok és táblák definícióját, valamint aktuális tartalmát megjelenítő nézethez. Az aktuális tartalmat megjelenítő táblanézet-fül alatt lehetőségünk van az adatbázis tartalom módosítására is – így például tesztriasztásokat is elhelyezhetünk az `alerts.status` táblában.

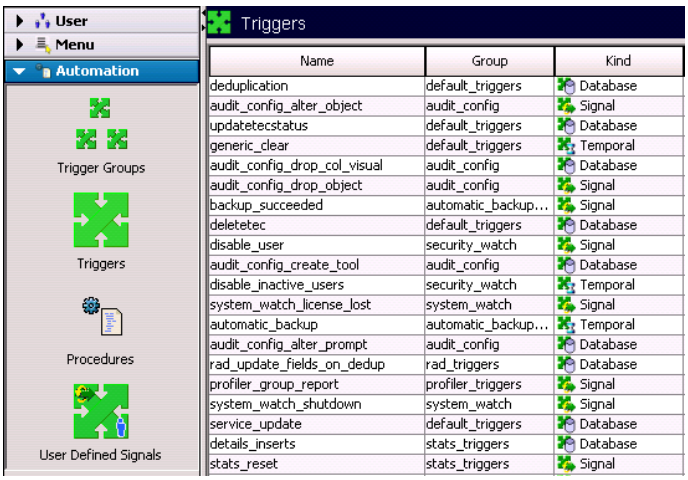
A triggerdefiníciók kezelése is az Administratorban történik; ezeket az `'Automation'` fül alatt találjuk.



7. ábra: Administrator - System/Databases, oszlopdefiníciós nézet



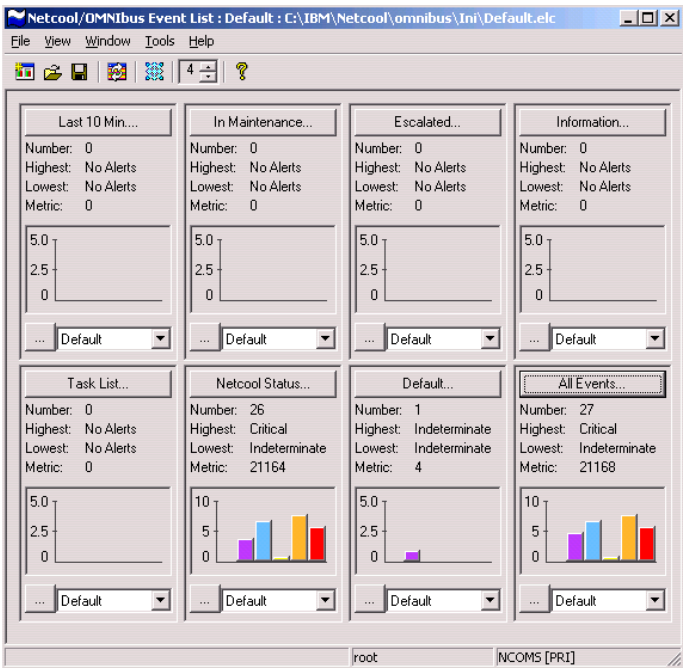
8. ábra - System/Databases, adatnézet és felugró menü a módosítási lehetőségekkel



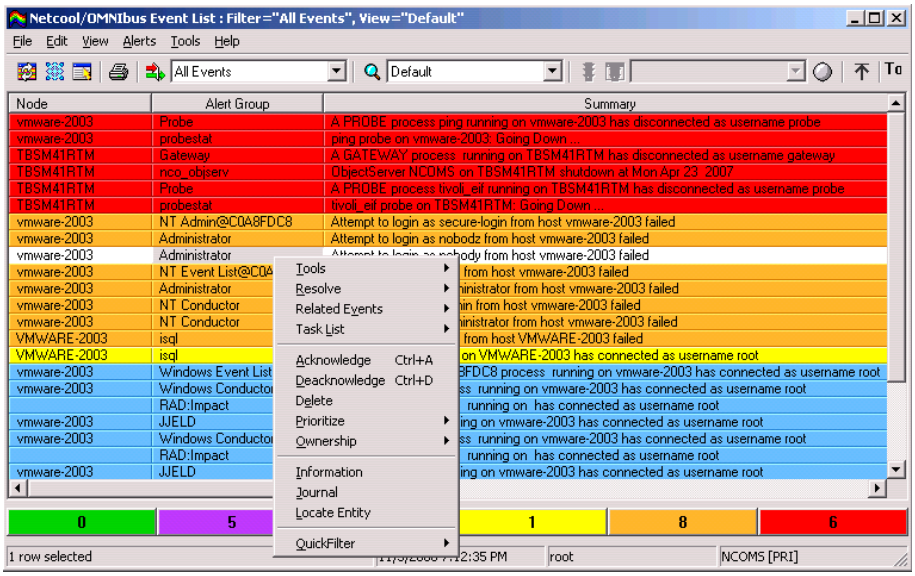
9. ábra: Administrator – Automation/Triggers

4.3 Az Event List alkalmazás

Az Event List alkalmazással lehet az ObjectServer-ben tárolt riasztásokat különböző módokon szűrve, rendezett és színezett eseménylista formájában megjeleníteni. Az alkalmazás bizonyos, általunk nem tárgyalt operátori akciókra is lehetőséget nyújt, mint például riasztások eszkalálása, egyszerű tudomásul vétele vagy „kezelés alatt" (maintenance) állapotba hozása. (Ezek az akciók egyébként a háttérben egyszerűen a riasztáshoz tartozó sor meghatározott mezőinek frissítését implementálják, potenciálisan triggereket is tüzelve.) Az egyes szűrési feltételeknek megfelelő eseménylisták a „..." gomb megnyomásával érhetőek el a kezdőképernyőről.



10. ábra: az Event List alapértelmezett filterkészlet-konfigurációja

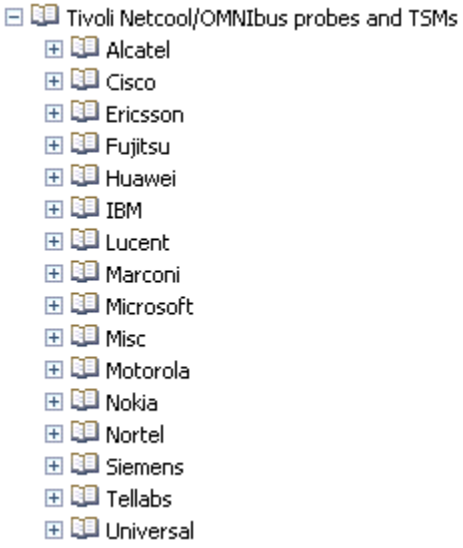


11. ábra: egy példa eseménylista az Event View alkalmazásban

4.4 Probe-ok (szondák)

Az OMNIbus adatbázisát eseményekkel ellátó komponenseket hívjuk szondáknak (probe-oknak). Definíció szerint egy szonda „kapcsolódik egy eseményforráshoz, detektálja és megszerzi az eseményadatokat, majd riasztások formájában továbbítja azokat az ObjectServer-nek”.

A szondák általában egy általános célú számítógépen szolgáltatásként futó folyamatok; többségük célspecifikus megoldás, melyek a Netcool SDK-jával készültek. Vannak azonban generikus, konfigurálható szondák is, illetve olyanok, melyek más rendszerfelügyeleti termékekből (mint például a Tivoli Monitoring vagy a Microsoft Operations Manager) vesznek át eseményeket.



12. ábra Az OMNIbus szonda-kategóriái

Néhány fontosabb univerzális szonda:

Szonda neve	Adatgyűjtés módja
Exec Probe	Fork-olt folyamat stdout-jának az értelmezése
FIFO Probe	Nevesített pipe-on kapott bemenet értelmezése
Generic ODBC Probe	Olvasás adatbázisból
Generic Log File Probe	Logállományból olvasás és értelmezés

Ping Probe	ICMP kérések elvégzése és kiértékelése
SNMP Probe	SNMP lekérdezés
Socket Probe	TCP/IP kiszolgáló socket-re kapott ASCII adatok értelmezése
Syslog Probe	UNIX rendszerlogok értelmezése
Syslogd Probe	Távoli UDP portra logolásra konfigurált syslogd-k üzeneteinek fogadása és értelmezése

A szondákhoz tartozó futtatható állományok, dinamikusan linkelt könyvtárak és konfigurációs állományok alapértelmezett esetben az `$OMNIHOME/probes/<arch>` könyvtárban találhatóak (Windows esetén: `%OMNIHOME%\probes\win32`). Minden szonda mellett jelen kell lennie egy `<szondanev>.props` és egy `<szondanev>.rules` állománynak. A `.props` állomány általános és típusspecifikus konfigurációs direktívákat tartalmaz; a `.rules` állomány egy speciális parancsnyelven azt adja meg, hogy az üzenetforrásból kinyert adatokból (típusonként előre megadott nevű változók) és a szondától lekérdezhető információból, mint pl. a hoszt neve (szintén előre ismert nevű változók) hogyan kell az ObjectServer felé elküldendő riasztást összeállítani. Mindkét állomány szöveges formátumú és az igényeknek megfelelően testreszabható.

4.4.1 Példa: a Ping Probe

A ping szondának (Windows esetén `%OMNIHOME%\probes\win32\nco_p_ping.exe`, illetve ha Windows szolgáltatásként installált, úgy alapértelmezetten az `NCOMTPingProbe` szolgáltatás) a `ping.props` és a `ping.rules` állományokon kívül még van egy `ping.file` konfigurációs állománya is, melyben a periodikusan ellenőrzendő hosztokat adhatjuk meg. A `ping.rules` állomány lényeges része:

```
[...]
# Available Elements:
#
###########
#$icmp_stats = icmp statistics of host
#$alias = host alias of host
#$host = host name of host
#$status = status of host
#$ip_address = IP address of host
###############################################################################
[...]
        @Identifier = $host + $status
        @Node = $host
        @NodeAlias = $alias
        @Manager  = "Ping Probe"
        @AlertGroup = "Ping"
        @Class = 100

        switch ($status)
        {
                case "unreachable" :
                        @Severity = "5"
```

```
                @Summary = @Node + " is not reachable"
                @Type = 1
        case "alive" :
                @Severity = "3"
                @Summary = @Node + " is now alive"
                @Type = 2
        case "noaddress" :
                @Severity = "2"
                @Summary = @Node + " has no address"
        case "removed" :
                @Severity = "5"
                @Summary = @Node + " has been removed"
        case "slow" :
                @Severity = "2"
                @Summary = @Node + " has not responded within trip time"
        case "newhost" :
                @Severity = "1"
                @Summary = @Node + " is a new host"
        case "responded" :
                @Severity = "0"
                @Summary = @Node + " has responded"
        default :
                @Summary = "Ping Probe Error details : " + $*
                @Severity = "3"
    }
}
```

A szabályállomány minden egyes alkalmazása esetén (azaz minden esetben, amikor a szonda belső logikája úgy dönt, hogy eseményt kellene küldeni) „bemenetként” néhány előre lekötött változót kap. Ezen változók nevét a ’$’ karakter prefixálja, az alkalmazható változók halmaza a dokumentációból vagy a szondával szállított minta szabályállományból derül ki (lásd a kikommentelt részt az idézett kód elején). ’@’ karakterrel prefixelve tudunk értéket adni az adott esemény beállítani szándékozott `alerts.status` mezőinek, illetve értékadás után értékükre hivatkozni a parancsállomány fennmaradó részében. Figyeljük meg az alapértelmezett eseménykorrelációs logika használatát a `Type` attribútum beállításával az ’`unreachable`’ és ’`alive`’ állapotokban!

4.4.2 Példa: a GLF (Generic Log File) Probe

A GLF szonda (Windows esetén `%OMNIHOME%\probes\win32\nco_p_glf.exe`, illetve ha szolgáltatásként installált, úgy alapértelmezetten az `NCOGLFProbe` szolgáltatás) log állományok megfigyelésére és azok tartalmából riasztások előállítására képes; a figyelendő logállomány a `glf.props` állományban adható meg (`LogFileName` tulajdonság)[3]. A `glf.props` állományban kulcsfontosságú, szonda típus specifikus tulajdonságok még a következők: `ValueSeparator` (értékelválasztó), `LineSeparator`, `QuoteCharacter`. A szonda az állományban megjelenő sorokat értelmezi külön-külön üzenetek adatforrásaként; az értékelválasztó karakter által elválasztott tokeneket pedig elhelyezi a szabályfeldolgozás `$FieldVal[n]` változóiban, ahol `n` egy ’`01`’-től

[3] Érdemes megjegyezni, hogy a tulajdonság-állományok alapértelmezés szerint a legtöbb tulajdonságot (vagy éppen mindet) az alapértelmezett értékkel összerendelve, de *kikommentelve* adják meg.

folytonosan inkrementált számláló minden üzenetre. Nézzük egy enyhén testreszabott parancsállomány érdemi részét:

```
[...]
#default settings:
      @Class = 7955
      @Identifier = @Manager + @AlertGroup + $Event
#default:
#      @Summary    = $*
#custom:
            @Summary = $FieldVal01
      @Node       = $Node
      @NodeAlias  = $DeviceType
      @Manager    = "glf"
          @FirstOccurrence = $Time
          @LastOccurrence = $Time
[...]
```

Mint látható, a `Summary` mező értékét lecseréltük a logállomány sorainak első tokenjére. A megfigyelt állományba beírt `'Hello_world!'` karakterlánc (+ enter, majd az állomány mentése) a következőképpen jelenik meg az Event List alkalmazásban:

Node	Alert Group	Summary	Last Occurrence	Count	Type	ExpireTime	Agent	Manager
		Hello_world!	12/31/1969 7:00:00...	1	Type Not Set	Not Set		glf

13. ábra: GLF szonda egyszerű testreszabásának tesztelése

5 Ellenőrző kérdések

- Mik a szolgáltatásbiztonság jellemző attribútumai?
- Oldja fel az SLA rövidítést és adjon a fogalomra rövid definíciót!
- Az SLA-k betartásának kötelezettsége milyen hatással van az informatikai infrastruktúrát támogató folyamatokra?
- Mi történik egy üzenettel a Netcool OMNIbus ObjectServeren és miért, ha a Severity mezőjének értéke 0-ra változik meg?
- Mi a különbség a Netcool OMNIbus ObjectServer alerts.status táblájának Severity és Type mezői között? A kettő egyszerre mely beépített mechanizmusban kap szerepet?
- Sorolja fel az OMNIbus néhány fontosabb univerzális szondáját és mindegyikre adja meg az adatgyűjtés módját!

6 Hivatkozások

[1] A Avizienis, J.C. Laprie, B. Randell, C. Landwehr, „*Basic Concepts and Taxonomy of Dependable and Secure Computing*", IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, 2004, IEEE Computer Society.

[2] openwebload: http://openwebload.sourceforge.net/

[3] IBM Tivoli Composite Application Manager for Transactions: http://www-01.ibm.com/software/tivoli/products/composite-application-mgr-transactions/

[4] IBM Tivoli SLA Advisor: http://www-01.ibm.com/software/tivoli/products/service-level-advisor/

[5] IBM Tivoli Business Service Manager (TBSM) Version 4, Release 2, *Exploring Tivoli Business Service Manager*. Dokumentumkód: GI11-8056-02; a dokumentum letölthető a következő URL-ről:

http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/topic/com.ibm.tivoli.itbsm.doc_4.2/tbsm42exploring.pdf

[6] IBM Tivoli Netcool/OMNIbus 7.2.1 online dokumentáció: http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/index.jsp?topic=/com.ibm.netcool_OMNIbus.doc_7.2.1/welcome.htm

[7] Netcool/OMNIbus alerts.status table reference: http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIbus.doc_7.2.1/common/reference/omn_ref_tab_alertsstatus.html

[8] Netcool/OMNIbus probe rules file syntax:

http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcool_OMNIbus.doc_7.2.1/probegtwy/reference/omn_prb_proberulesfilesyntax.html