2012. Szeptember

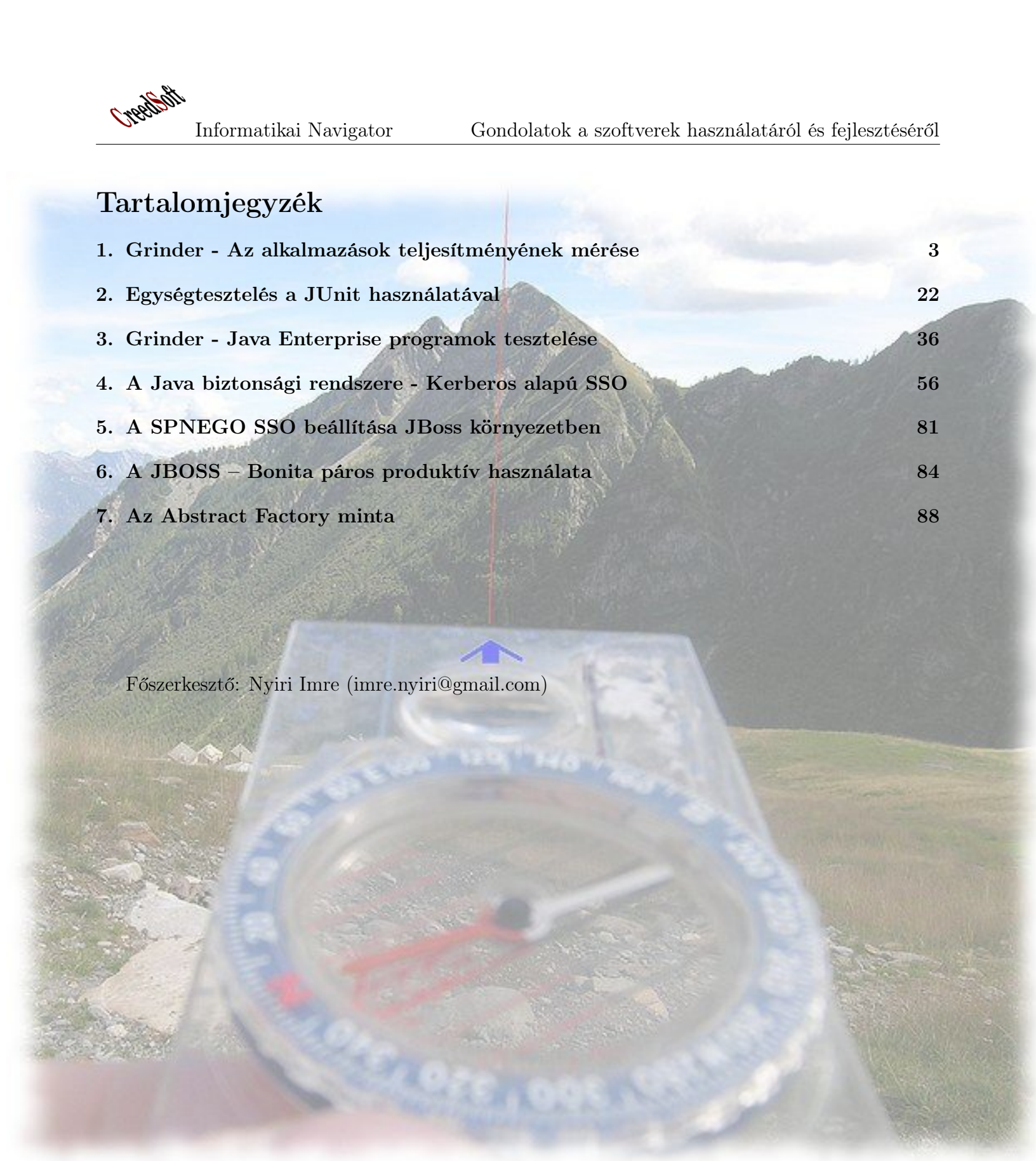
Informatikai Navigátor

Gondolatok a szoftverek használatáról és fejlesztéséről

CreedSoft

6. szám

Tartalomjegyzék



Főszerkesztő: Nyiri Imre (imre.nyiri@gmail.com)

1. Grinder - Az alkalmazások teljesítményének mérése

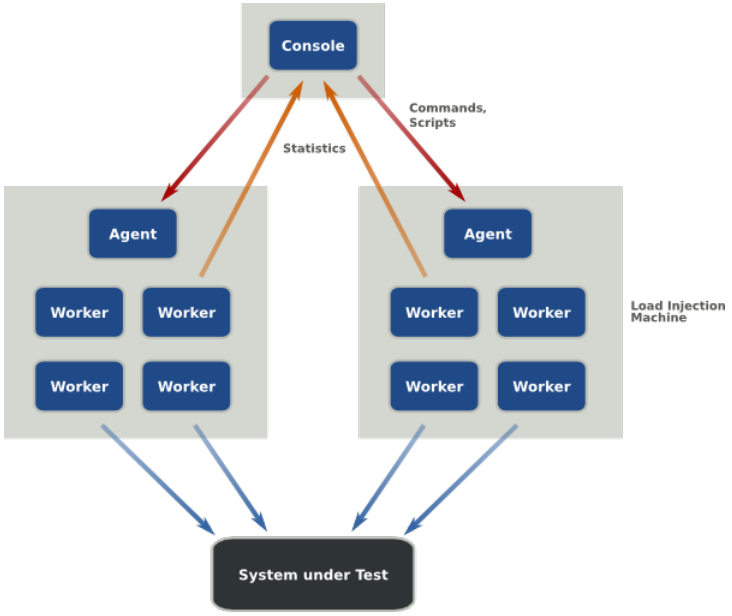
Írásunk a Grinderről, azaz a darálóról szól. Ez egy olyan eszköz, ami lehetővé teszi a terheléses (stressz) alkalmazás tesztek elvégzését. Elsősorban a web alkalmazások tesztelésére lett kitalálva, de ugyanúgy alkalmasnak bizonyult az adatbázisok, java JMS queue-k és egyéb komponensek terhelhetőségének vizsgálatára, amit a beépített Python (Jython) motor tesz lehetővé. A project webhelye: `http://grinder.sourceforge.net/`.

A *Grinder* eredetileg egy terheléstesztelő eszközként készült, azonban amióta Python (Jython) scriptben készíthetjük el a teszteseteket, jól alkalmazható az alkalmazások funkcionális helyességének teszteléséhez is. Az eszközzel megállapítható az alkalmazás kiszolgáló képességének maximuma (capacity=kapacitása), illetve a *Stress Testing* funkcióval ellenőrizni tudjuk, hogy a produktív üzemre tervezett áteresztő képesség biztosított-e. A Grinder legfontosabb lehetőségei:

- A böngészők működésének szimulációja egy felvett interakció ismételt lejátszásával
- Webservicek tesztelése
- JMS, JDBC és egyéb JEE kommunikációk tesztelése
- Különféle internetes protokollok (POP3, SMTP, FTP, LDAP, ...) feletti kommunikációk tesztelése

A Grinder felépítése

A *Grinder* architekturális felépítését az 1.1. ábráról tanulmányozhatjuk. Látható, hogy általában több process fut egyszerre. A konzol process tartja a kapcsolatot az agent (ügynök) processekkel, amik különböző fizikai számítógépen futhatnak. Ez a kapcsolat kétirányú, mert a konzolról parancsok adhatóak az agentek felé, ugyanakkor az ügynökök folyamatosan továbbítják a mérési eredményeket a központi konzol számára. Az agentek 1 vagy több munkafolyamatot (*Worker* process) indíthatnak, ezeken belül pedig szálak (*Worker Threads*) futnak, amik végső soron a Python nyelven megírt teszt scripteket futtatják.



1.1. ábra. A Grinder felépítése

Az agentek tehát a Worker-eket menedzselik, elindítják és leállítják azokat, illetve fenntartják a megfelelő futási környezetüket. Mindegyik *Worker* folyamat naplót vezet, aminek a file neve a következő konvenció szerint képződik: *host-n.log*, ahol a host a gép neve, az *n* pedig a *Worker* process sorszáma. A tesztelés közben keletkezett adatok a *host-n-data.log* file-okban képződnek, ahol ugyanez a névkonvenció érvé-

nyesül. A file formátuma CSV, emiatt könnyen lehet beolvasni azt egy táblázatkezelőbe és ezzel különféle utóelemzések végezhetőek el. Egy tesztelés lehetséges kimenetele SUCCESS (sikerült) vagy ERROR (hiba történt) lehet.

A Grinder egyes elemeinek indítása

A szoftver háromféle indítható programmal rendelkezik:

- A konzol process, ahol adminisztráljuk és monitorozzuk a munkát.
- Az Agent processek, amik a konzolhoz kapcsolódnak.
- Egy TCP/HTTP proxy, ami a tesztfuttatáshoz nem szükséges, szerepét a későbbiekben ismertetjük.

A következőkben néhány ajánlott *bash* parancsállományt javaslunk, ezekkel érdemes elindítani az egyes processeket. A *setGrinderEnv.sh* a környezeti változók beállítását végzi. A *GRINDERPATH* az a hely, ahova a Grinder-t kicsomagoltuk. A *GRINDERPROPERTIES* a konfigurációs file helye.

```
// setGrinderEnv.sh

#!/bin/bash

GRINDERPATH=/home/tanulas/grinder/grinder➥
    -3.7.1
GRINDERPROPERTIES=/home/tanulas/grinder/➥
    grinder.properties
CLASSPATH=$GRINDERPATH/lib/grinder.jar:➥
    $CLASSPATH
JAVA_HOME=/usr/lib/jvm/java-6-sun
PATH=$JAVA_HOME/bin:$PATH
export CLASSPATH PATH GRINDERPROPERTIES
```

Sorrendben mindig a konzolt kell először elindítani, hiszen az agentek erre fognak csatlakozni. A *startConsole.sh* script a konzol javasolt indítását automatizálja.

```
// startConsole.sh

#!/bin/bash

. ./setGrinderEnv.sh
java -classpath $CLASSPATH net.grinder.Console
```

Amikor a konzol elindult (ez egy GUI program) startoltathatjuk az agent-eket is, minimum 1 darabot. Ez természetesen ugyanazon a gépen is lehet, mint a konzol, a *startAgent.sh* mutatja az ajánlott indítást.

```
// startAgent.sh

#!/bin/bash
. ./setGrinderEnv.sh
java -classpath $CLASSPATH net.grinder.Grinder➥
    $GRINDERPROPERTIES
```

A proxy process-t nem közvetlenül a tesztelésnél használjuk, ezért annak indítását és használatát egy külön pontban ismertetjük.

A Grinder konfigurációja

A konfigurációt a *grinder.properties* tartalmazza, aminek lehetséges részeit mutatjuk be a következőkben.

grinder.processes

Az *Agent* által elindítandó dolgozó (*Worker*) folyamatok (process) száma. Ezzel lehet utánozni a többfelhasználós működést a tesztelés során. Az alapértelmezett érték: *1.*

grinder.threads

A *Worker* folyamathoz itt lehet meghatározni, hogy mennyi szálon fusson. A böngészők is több szálon szokták az erőforrásokat letölteni, így ez a működés a tökéletesebb böngésző szimulációban segít. Az alapértelmezett érték: *1.*

grinder.runs

Itt lehet megadni, hogy a Python teszt scriptet hány alkalommal futtasson le egy tesztelő szál. Az alapértelmezett érték: *1.* A *0* azt jelenti, hogy a script örökké fut, azaz lefutása után ismételten elindításra kerül. Ezt akkor érdemes használni, amikor a *Grinder* konzolt folyamatosan figyelni szeretnénk.

grinder.processIncrement

Amikor ezt megadjuk, akkor a *Grinder* nem indítja el az összes lehetséges munkavégző folyamatot (emlékezzünk, ez a *grinder.processes* property-ben van megadva), hanem csak egy *grinder.initialProcesses*-ben megadott darabszámot, amit a *grinder.processIncrementInterval* (alapértelmezés: 60000 ms) időközönként növel, amíg el nem érjük a definiált maximális munkafolyamat számot. Amennyiben nem adjuk meg, úgy alapértelmezetten az összes *Worker* process az elején elindul.

grinder.processIncrementInterval

Szerepét a *grinder.processIncrement* résznél ismertettük.

grinder.initialProcesses

Szerepét a *grinder.processIncrement* résznél ismertettük.

grinder.duration

Ezredmásodpercben megadhatjuk, hogy a *Worker* processek mennyi ideig futhatnak, az alapértelmezés az örökké tartó futás. Ennek hatása – amennyiben megadjuk – együtt érvényes a már ismertetett *grinder.runs* konfigurációs lehetőséggel, mely esetben a *Worker* process megszűnik, ha eléri a megadott időtartamot (*duration*) vagy lefutott annyiszor, amennyit a futások száma (*runs*) előír.

grinder.script

Az a python file név (alapértelmezés: *grinder.py*), amit a *Worker* processek futtatnak.

grinder.logDirectory

Az a könyvtár, ahova a *Grinder* a logfile-okat helyezi. Amennyiben még nem létezik, úgy automatikusan létrejön.

grinder.hostID

Amennyiben megadjuk, úgy nem az agent gép host neve kerül a naplóba, hanem az itt megadott string.

grinder.consoleHost

A *Grinder* központi konzolja ezen a host-on fut, azaz erre az IP címre kell a *Worker* processeknek (Agent-ek) kapcsolódniuk. Az alapértelmezés a helyi gép összes hálózati interface-e.

grinder.consolePort

A *grinder.consoleHost* property mellé tartozó TCP/IP port, az alapértelmezés a 6372.

grinder.useConsole

Az alapértelmezés a *true*, de amennyiben ezt *false*-ra állítjuk, úgy a *Worker* process nem kapcsolódik a konzolhoz, így a konzolon keresztül elérhető szolgáltatások sem vehetők ebben az esetben igénybe.

grinder.reportToConsole.interval

Az a periódusidő (alapértelmezés: 500 ms), ahogy a *Worker* process-ek információt küldenek a *Grinder* konzol felé.

grinder.initialSleepTime

Az maximális idő ezredmásodpercben, amennyit minden futási szál (*thread*) vár mielőtt elindulna (alvási idő). Az alapértelmezett érték a 0 ms. Itt egy olyan véletlen-szám eloszlás van, aminek ez a maximuma, de 0 és e között bármekkora lehet ez az érték.

grinder.sleepTimeFactor

Az alapértelmezett értéke 1. Ez egy olyan szorzótényező, amivel a tényleges alvásidő kalkulálódik, azaz például 0.1 érték esetén 10-szer gyorsabban indul el a *Worker* process.

grinder.sleepTimeVariation

A *Grinder* alvási ideje a normális eloszlást fogja követni. Egy példán lehet legkönnyebben megérteni. Legyen az *grinder.initialSleepTime=1000* ms és a *grinder.sleepTimeVariation=0.1*, azaz 10%=100 ms. Ekkor ez meghatározza az 1000-100 és 1000+100, azaz a [900, 1100] időintervallumot. Definíció szerint ekkor a futások 99.75 %-a ebbe az intervallumba eső ideig fog aludni az elindulásuk előtt. Ezen paraméter alapértelmezett értéke: 0.2.

grinder.reportTimesToConsole

Az alapértelmezett érték a *true*, de *false* esetén az időzítési (*timing*) információkat a *Grinder* nem küldi el a konzol felé.

grinder.jvm

Egy eltérő JVM is megadható, ekkor a *java* parancs helyett a másik JVM futási nevét kell megadni. Természetesen a *PATH* környezeti változó helyes beállítása lényeges. Az alapértelmezett érték: java.

grinder.jvm.classpath

Az itt megadott *PATH*-ok a java *CLASSPATH* részei lesznek.

grinder.jvm.arguments

További parancssori argumentumok a teszteket futtató JVM számára. Például itt adhatunk több memóriát a Java Virtuális gépnek.

A Grinder konzol megismerése

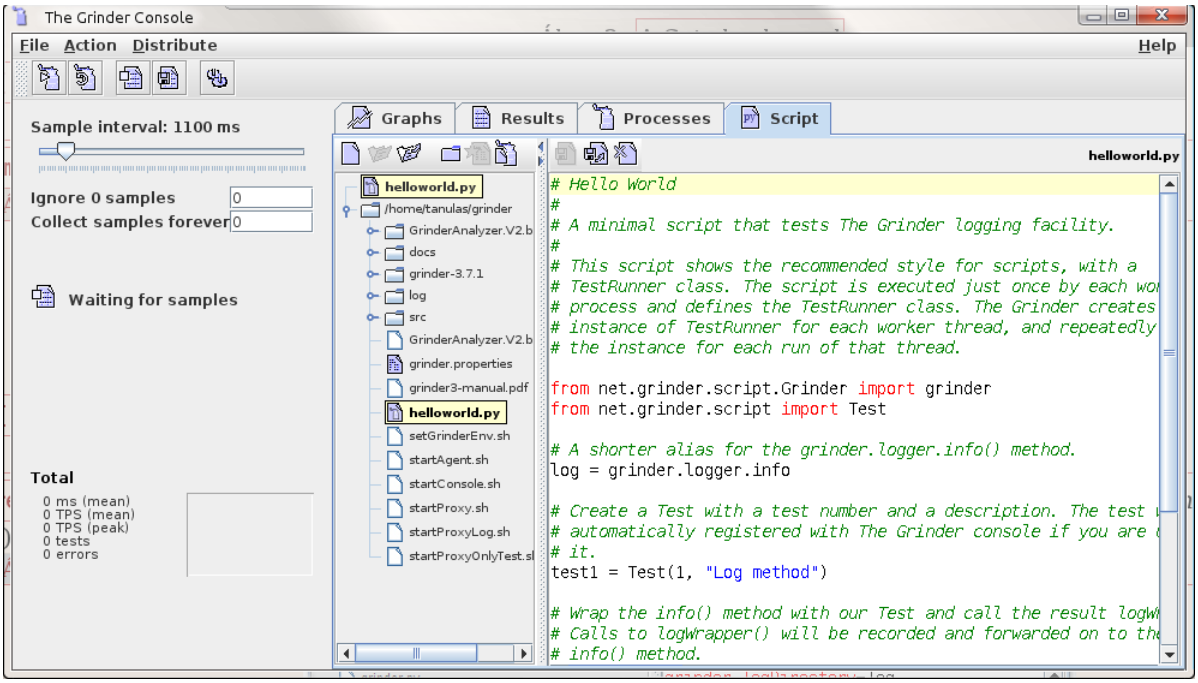
A 1.2. ábra a Grinder konzol képét mutatja, ahol a *Graphs* (grafikus áttekintés), *Results* (eredmények), *Processes* (beállított folyamatok), *Script* (python program, ami éppen be van töltve) fülek jelentik a konzolon végezhető főbb feladatok helyeit.

A folyamatok kezelése

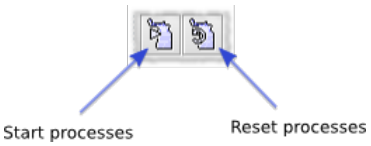
A folyamatkezelés a tesztek elindítását (Start processes), alaphelyzetbe hozását (Reset processes) és leállítását (Stop processes) jelenti, amely funkció az Action menüből vagy az 1.4. ábrán látható ikonok révén érhető el. Ez a háttérben az előzetesen elindított és a konzolra beregisztrált Grinder (agent) processek felé történő parancsküldésekkel valósul meg, amiatt ezek a funkciók nem érhetőek el („szürkék"), amikor a konzolra nincs egyetlen agent se kapcsolódva. A beregisztrált Grinder Worker processek a *Processes* fülön tekinthetőek meg és a következő állapotok egyikében lehetnek:

- *Initiated*: várakozás egy konzol parancsra
- *Running*: a process éppen tesztet futtat, eközben küldi a riport adatokat a konzol felé
- *Finished*: várakozás egy konzol parancsra, mert lefutott

A *Start processes* parancs mindig *Running* állapotba teszi a Workert, illetve amennyiben az már fut, úgy figyelmen kívül marad. A *Reset processes* parancs leállítja a Workert, újraolvassa a konfigurációt és ismét elindítja a tesztelést. A *Stop processes* parancs úgy állítja le az összes agent-et, hogy azoknak az operációs rendszerbeli futásuk is véget ér.



1.2. ábra. A Grinder konzol



1.3. ábra. Grinder folyamatok kezelése

A A 1.2. ábra bal oldali részén lehet szabályozni (ezt a *grinder.reportToConsole.interval* property-ben is megadhattuk), hogy a konzol mennyi időnként vegyen át mérési mintát a *Worker*-ektől. A *Script* fül mögött egy szövegszerkesztő van, ahova a Python nyelvű teszteket is betölthetjük, a *Grinder* ezt fogja futtatni. Az ábrán éppen a később ismertetendő *Hello World* tesztprogram egy részlete látszik. A *Graphs* és *Results* fülek a tesztekről mutatnak információt (1.4 és 1.5 ábrák). Fontos mértékegység a *TPS* (Test Per Secundum), ami azt jelenti, hogy 1 másodperc alatt mennyi tesztfutás történt. A piros oszlopdiagram azt mutatja, hogy az utolsó 25 tesztnél mennyi volt a mért érték. Az átlag (*mean*) esetünkben 38000 TPS, míg a csúcsérték (*peak*) 92700 TPS volt. Az diagram Y tengelye úgy van skálázva, hogy a teteje mindig a TPS csúcsértéket jelenti. A Hello World egyszerű program, így 1.174.876 darab tesztet végeztünk el a mérés kb. 32 másodperce alatt. A 0.0235 ms 1 darab teszt átlagos futási ideje. A bal oldalon látható 32800 TPS az utolsó mért érték.

A *Results* fül a következő mérési adatokat mutatja, ahogy azt egy másik mérésre a 1.5. ábra tartalmazza is:

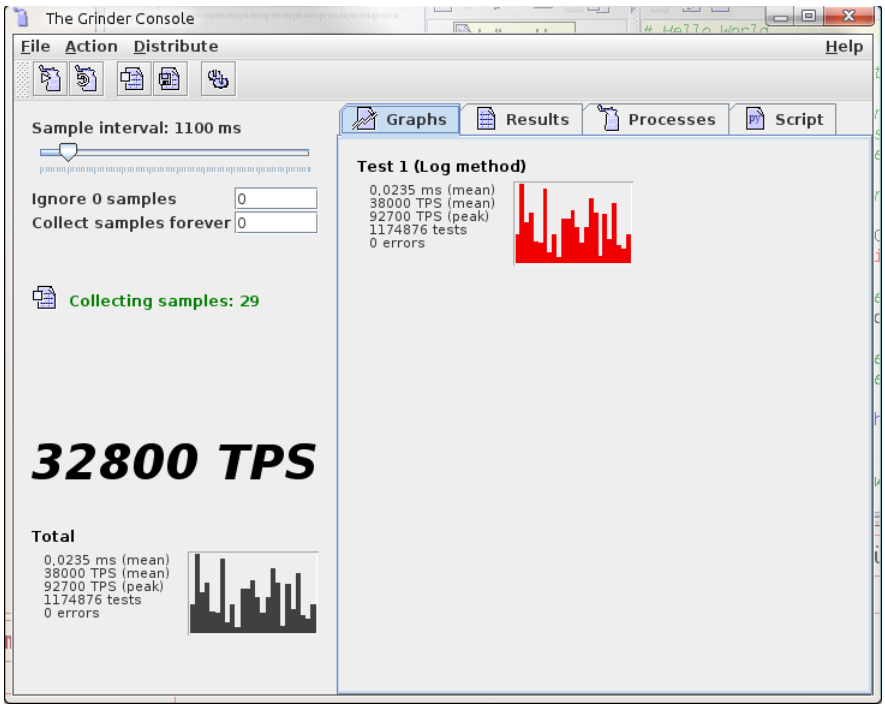
- *Test*: A teszt azonosító sorszáma, amit a teszt script írásakor a *Test* class első paramétereként adtunk meg.

- *Description*: A *Test* class 2. paramétere, ami egy string. A generált http proxy alapú script-ek itt adják meg a lekért erőforrás nevét és módját, például: *Test(14000, 'GET index.jsp')*. Ez 1400 értékű tesztazonosítót is mutat, aminek a jelentését az előző pontban magyaráztuk el.

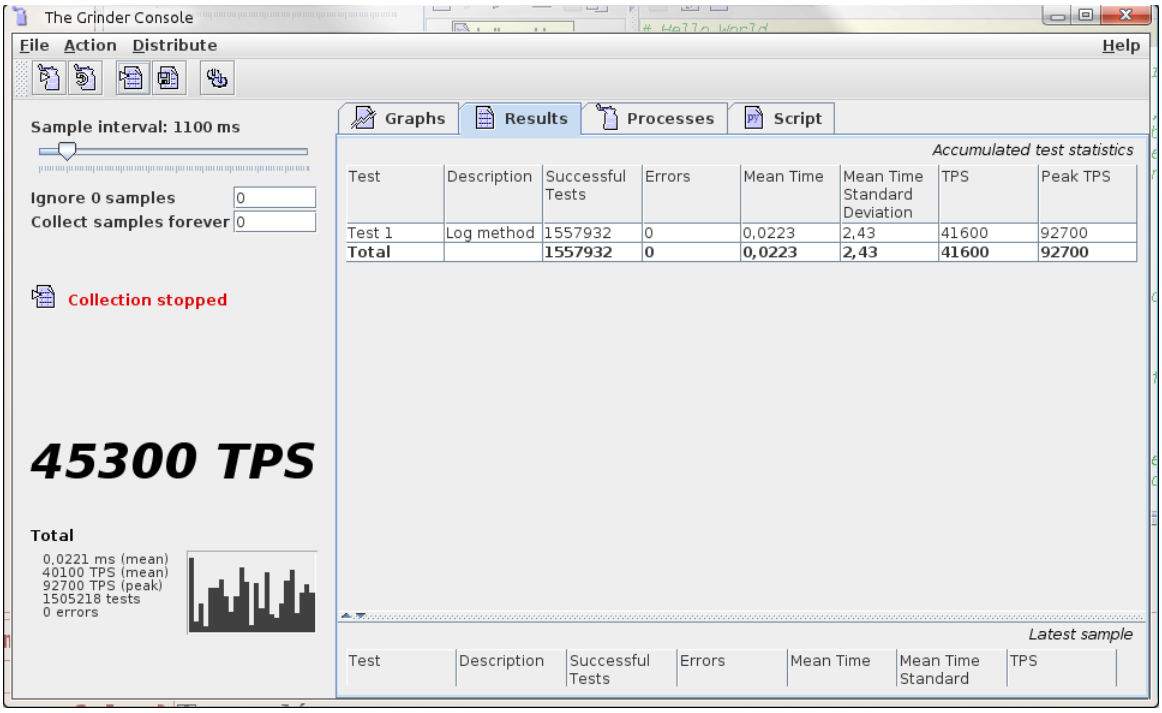
- *Successful Tests*: A sikeresen elvégzett tesztek száma (a példánkban 1.557.942 db).
- *Errors*: A hibára futott tesztek száma (példánkban ez most 0 db)
- *Mean Time*: Átlagos tesztfutási idő (példánkban 0.0223 ms). Ez http kérés esetén természetesen magában foglalja a válasz teljes beérkezését is.
- *Mean Time Standard Deviation*: A tesztfutási idők standard szórása (példánkban: 2.43).
- *TPS*: Itt átlagos tranzakciószámot jelent másodpercenként (most 41600 db/s)
- *Peak TPS*: Az eddigi legnagyobb mért TPS (itt 92700 db/s)

Amennyi HTTP alapú mérést végzünk, úgy a következő mérési adatok is a rendelkezésünkre állnak:

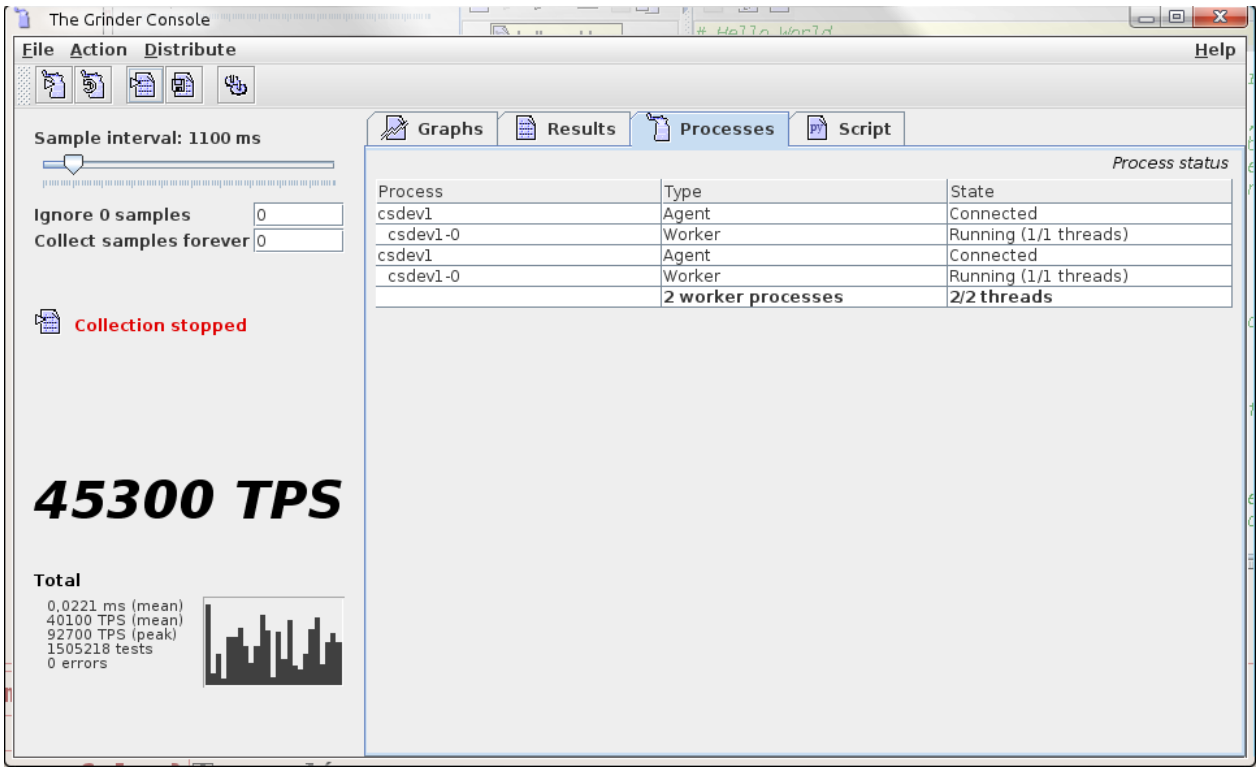
- *Mean Response Length*: A http válaszok byte-ban mért átlagos mérete.
- *Response Bytes per Second*: A http válaszok átlagos byte/s sebessége.
- *Response Errors*: A http válaszok során ennyi teszt futott hibára, például 404 vagy 500-as http hibakód miatt.
- *Mean Time to Resolve Host*: A DNS névfeloldás átlagos időszükséglete ezredmásodpercben.
- *Mean Time to Establish Connection*: A TCP kapcsolat felépítésének átlagos időszükséglete ezredmásodpercben.
- *Mean Time to First Byte*: A http válasz első byte-jának átlagos megérkezési ideje.



1.4. ábra. A Grinder tesztfuttatás közben - Graph fül



1.5. ábra. A Grinder tesztfuttatás közben - Result fül



1.6. ábra. A Grinder tesztfuttatás közben - Processes fül

A 1.6. ábráról, a *Processes* fülről a kezelt tesztelő folyamatokról tudhatjuk meg a legfontosabbakat. Korábban már elmagyaráztuk ennek a hierarchikus felépítését. Az *Agent*-ek több példányban indíthatóak, különböző gépekről is. Esetünkben a *csdev1* nevű gépről 2 példányban adtuk ki a *startAgent.sh* parancsot. A *grinder.properties* file-ban a *grinder.processes = 1* és *grinder.threads = 1* a beállítás, amiatt minden Agenthez 1 db Worker, ahhoz pedig csak 1 db futási szál tartozik.

Tesztelés

Ennyi ismeret után kezdjük el használni a Grindert! Ahogy a kiadványunk borítóján lévő mesterünktől (*Dennis M. Ritchie*, a C nyelv megalkotója) tanultuk, ez most is legyen a Hello World tesztprogram (1-1. programlista)! Az eddigi képernyőkön ennek a tesztnek a futási képeit használtuk fel a magyarázatokhoz. A 7. sor egy Python lehetőséget használ, történetesen *log* néven egy alias nevet rendel a *grinder.logger.info()* metódushoz, így az INFO szintű loggoláshoz a *log(...)* kifejezés is használható lesz. A 13. sorban egy *test1* nevű tesztelésre képes, *Test* osztálybeli objektumot hozunk létre, emlékezzünk, hogy a konstruktor paramétereit láttuk a Grinder konzolon. A 18. sor kissé trükkös, de ez biztosítja, hogy a tesztfuttató környezet és a *helloworld.py* programunk között egyértelmű legyen, hogy mit kell futtatni majd a 22-26 sorban megadott *TestRunner* osztályban. A létrehozott *logWrapper* nevű objektum egyrészt a *teszt1.wrap()* alias neve, ugyanakkor a *Test.wrap()* implementációja pedig a *log*, azaz valójában a *grinder.logger.info()* metódus. Mindez úgy, hogy ebben a játékban a *logWrapper* paraméterezési felülete ugyanaz marad, mint az eredeti *log()*, azaz *info()* metódus felülete. A *TestRunner __call__(self)* metódusa kötött, ezzel tud kommunikálni a tesztfuttató szál. A tesztelés leállítása után a log könyvtárban 2 filet érdemes megnézni (a nevük névkonvencióját már ismertettük):

- *csdev1-0.log*: Ide naplóz a tesztfuttatás, ilyen sorokból áll:

```
2012-03-03 08:10:08,192 INFO csdev1-0 ➥
    thread-1 [ run-0, test-1 ]: Hello ➥
    World)
```

- *csdev1-0-data.log*: Ide kerülnek a tesztmérések

A mért adatok pedig ilyenekből (itt a Workerek száma 2, a Thread-ek száma 3 volt méréskor):

```
Thread, Run, Test, Start time (ms since Epoch)➥
    , Test time, Errors
2, 0, 1, 1330762208189, 1, 0
1, 0, 1, 1330762208191, 1, 0
1, 1, 1, 1330762208192, 0, 0
1, 2, 1, 1330762208193, 0, 0
1, 3, 1, 1330762208193, 1, 0
1, 4, 1, 1330762208194, 0, 0
1, 5, 1, 1330762208194, 0, 0
1, 6, 1, 1330762208195, 0, 0
1, 7, 1, 1330762208195, 1, 0
1, 8, 1, 1330762208196, 0, 0
1, 9, 1, 1330762208197, 0, 0
1, 10, 1, 1330762208198, 0, 0
1, 11, 1, 1330762208199, 1, 0
2, 1, 1, 1330762208200, 0, 0
2, 2, 1, 1330762208201, 0, 0
2, 3, 1, 1330762208202, 1, 0
1, 12, 1, 1330762208201, 3, 0
2, 4, 1, 1330762208205, 0, 0
0, 0, 1, 1330762208205, 0, 0
2, 5, 1, 1330762208206, 0, 0
0, 1, 1, 1330762208206, 0, 0
1, 13, 1, 1330762208205, 0, 0
```

Vegyük észre, hogy ez a fájlformátum elsősorban az ismertebb táblázatkezelőkben való feldolgozhatóságot támogatja, de ebben a cikkben ismertetjük a *Grinder Analyzer* eszközt is, ami kiemelkedő minőségben támogatja a mért adatok elemzését. Látható, hogy a teszt python scriptek lényegében bármilyen tesztesetet implementálhatnak, ami azért jelent nagyon hatékony megoldást, mert a futtató Jython környezet lényegében bármilyen technológia elérését biztosítja. A következő példánkban már egy egyszerű HTTP GET lekérés tesztelését fogjuk bemutatni.

```
1  // 1-1. programlista: Hello World
2
3  from net.grinder.script.Grinder import grinder
4  from net.grinder.script import Test
5
6  # A shorter alias for the grinder.logger.info() method.
7  log = grinder.logger.info
8
9  # Create a Test with a test number and a description. The test will be
10 # automatically registered with The Grinder console if you are using
11 # it.
12 # teszt number és leírás
13 test1 = Test(1, "Log method")
14
15 # Wrap the info() method with our Test and call the result logWrapper.
16 # Calls to logWrapper() will be recorded and forwarded on to the real
17 # info() method.
18 logWrapper = test1.wrap(log)
19
20 # A TestRunner instance is created for each thread. It can be used to
21 # store thread-specific data.
22 class TestRunner:
23
24     # This method is called for every run.
25     def __call__(self):
26         logWrapper("Hello World")
```

Egy HTTP GET teszt

Az 1-2. programlista egy olyan tesztet valósít meg, ahol a `http://index.hu` URL-t kérdezzük le 25 másodpercen keresztül (a pontos lekérdezési időt a *„csdev1-0.log file vége futási kép”* 9. sorában láthatjuk, ez 24969 ms). A tesztprogram felépítése a Hello World-höz hasonlít, a 7-8 sorban itt is az ismertetett „aliasos” becsomagolása történik a *HTTPRequest()* hívásnak. Ezen metódus paramétere egy URL, amiatt a *TestRunner* osztályon belül alkalmazhatjuk a 12. sornál lévő ugyanilyen paraméterezésű hívást, ami a lekért HTML tartalmat a *result* változóba menti. A 16. sor már nem lenne feltétlenül szükséges ehhez a teszthez, csak lementi a HTML válaszokat egy-egy file-ba *csdev1-0-page-nnn.html* néven, ahol *nnn* a tesztlépés sorszáma. A teszt futtatása előtt nem szabad elfelejteni, hogy a *grinder.properties* fileban a *grinder.script* = *grinder-3.7.1/examples/http.py* beállítás legyen! Ennyi előzmény után kb. 25 másodpercig futtattuk a tesztet, aminek az eredményéről, illetve annak legfontosabb részeiről a

- A csdev1-0.log file eleje futási kép
- A csdev1-0.log file vége futási kép
- csdev1-0-data.log file eleje futási kép

tájékoztatja a kedves olvasót. Szeretnénk kiemelni, hogy a csdev1-0.log file végén megtalálhatóak azok az extra adatok is, amiket úgy emeltünk ki, hogy HTTP alapú tesztek esetén kerülnek be a logba. A csdev1-0-data.log pedig a későbbi statisztikákhoz segít. A következő fejezetben egy izgalmas dolgot csinálunk, ugyanis egy Grinder elemzőnek adjuk át ezeket az adatokat.

```
1  // 1-2. programlista: Egy http GET teszt
2
3  from net.grinder.script.Grinder import grinder
4  from net.grinder.script import Test
5  from net.grinder.plugin.http import HTTPRequest
6
7  test1 = Test(1, "Request resource")
8  request1 = test1.wrap(HTTPRequest())
9
10 class TestRunner:
11     def __call__(self):
12         result = request1.GET("http://index.hu")
13
14         # result is a HTTPClient.HTTPResult. We get the message body
15         # using the getText() method.
16         writeToFile(result.text)
17
18 # Utility method that writes the given string to a uniquely named file.
19 def writeToFile(text):
20     filename = "%s-page-%d.html" % (grinder.processName, grinder.runNumber)
21
22     file = open(filename, "w")
23     print >> file, text
24     file.close()
```

```
1  // A csdev1-0.log file eleje futási kép:
2  // ------------------------------------
3  2012-03-03 19:19:01,171 INFO  csdev1-0 : The Grinder version 3.7.1
4  2012-03-03 19:19:01,177 INFO  csdev1-0 : Java(TM) SE Runtime Environment 1.6.0_26-b03: Java HotSpot(TM) Server ➥
       VM (20.1-b02, mixed mode) on Linux i386 3.0.0-8-generic
5  2012-03-03 19:19:01,182 INFO  csdev1-0 : time zone is GMT (+0000)
6  2012-03-03 19:19:01,343 INFO  csdev1-0 : worker process 0 of agent number 0
7  2012-03-03 19:19:01,527 INFO  csdev1-0 : instrumentation agents: traditional Jython instrumenter; byte code ➥
       transforming instrumenter for Java
8  2012-03-03 19:19:04,069 INFO  csdev1-0 : registered plug-in net.grinder.plugin.http.HTTPPlugin
9  2012-03-03 19:19:04,089 INFO  csdev1-0 : running "grinder-3.7.1/examples/http.py" using Jython 2.2.1
10 2012-03-03 19:19:04,132 INFO  csdev1-0 thread-0: starting, will run forever
11 2012-03-03 19:19:04,132 INFO  csdev1-0 : start time is 1330802344133 ms since Epoch
12 2012-03-03 19:19:04,487 INFO  csdev1-0 thread-0 [ run-0, test-1 ]: http://index.hu/ -> 200 OK, 156145 bytes
13 2012-03-03 19:19:04,750 INFO  csdev1-0 thread-0 [ run-1, test-1 ]: http://index.hu/ -> 200 OK, 156145 bytes
14 2012-03-03 19:19:04,987 INFO  csdev1-0 thread-0 [ run-2, test-1 ]: http://index.hu/ -> 200 OK, 156145 bytes
```

```
1  // A csdev1-0.log file vége futási kép:
2  // ------------------------------------
3  2012-03-03 19:19:28,715 INFO  csdev1-0 thread-0 [ run-114, test-1 ]: http://index.hu/ -> 200 OK, 156145 bytes
4  2012-03-03 19:19:28,927 INFO  csdev1-0 thread-0 [ run-115, test-1 ]: http://index.hu/ -> 200 OK, 156145 bytes
5  2012-03-03 19:19:29,036 INFO  csdev1-0 : received a stop message
6  2012-03-03 19:19:29,087 INFO  csdev1-0 thread-0 [ run-116, test-1 ]: http://index.hu/ -> 200 OK, 156145 bytes
7  2012-03-03 19:19:29,098 INFO  csdev1-0 thread-0 [ run-117 ]: shut down
8  2012-03-03 19:19:29,098 INFO  csdev1-0 thread-0: finished 117 runs
9  2012-03-03 19:19:29,101 INFO  csdev1-0 : elapsed time is 24969 ms
10 2012-03-03 19:19:29,102 INFO  csdev1-0 : Final statistics for this process:
11 2012-03-03 19:19:29,121 INFO  csdev1-0 :
12             Tests         Errors        Mean Test     Test Time     TPS           Mean          Response      Response➥
                                  Mean time to Mean time to Mean time to
13                                Time (ms)     Standard                  response      bytes per     errors ➥
                                                resolve host establish    first byte
14                                              Deviation                 length        second ➥
                                                                                        connection
15                                              (ms)
16
17 Test 1        117           0             194,91        46,97         4,69          156145,00     731665,87     0 ➥
                    0,76          21,32         46,85          "Request resour
18 ce"
19
20 Totals        117           0             194,91        46,97         4,69          156145,00     731665,87     0 ➥
                    0,76          21,32         46,85
```

```
1  // csdev1-0-data.log file eleje futási kép:
2  // ----------------------------------------
3  Thread, Run, Test, Start time (ms since Epoch), Test time, Errors, HTTP response code, HTTP response length, ➥
       HTTP response errors, Time to resolve host, Time to establish c
4  onnection, Time to first byte
5  0, 0, 1, 1330802344147, 341, 0, 200, 156145, 0, 89, 140, 178
6  0, 1, 1, 1330802344523, 228, 0, 200, 156145, 0, 0, 17, 53
7  0, 2, 1, 1330802344766, 221, 0, 200, 156145, 0, 0, 13, 49
8  0, 3, 1, 1330802345003, 135, 0, 200, 156145, 0, 0, 13, 30
9  0, 4, 1, 1330802345160, 213, 0, 200, 156145, 0, 0, 48, 71
10 0, 5, 1, 1330802345393, 177, 0, 200, 156145, 0, 0, 32, 52
11 0, 6, 1, 1330802345587, 163, 0, 200, 156145, 0, 0, 18, 42
12 0, 7, 1, 1330802345766, 217, 0, 200, 156145, 0, 0, 18, 45
13 0, 8, 1, 1330802346003, 160, 0, 200, 156145, 0, 0, 12, 31
14 0, 9, 1, 1330802346180, 258, 0, 200, 156145, 0, 0, 13, 67
15 0, 10, 1, 1330802346459, 199, 0, 200, 156145, 0, 0, 34, 50
16 0, 11, 1, 1330802346675, 249, 0, 200, 156145, 0, 0, 31, 48
```

A Grinder Analyzer eszköz

A *Grinder Analyzer* (`http://track.sourceforge.net/analyzer.html`) segítségével a mért tesztadatok további feldolgozása valósítható meg, ami a generált log adatok feldolgozására épül. Az eredmény teljesítmény grafikonok (*performance graphs*) formájában jelennek meg, amelyek a következő elemeket tartalmazzák:

- válaszidők (response time)
- a tranzakciók sebessége (transactions per second)
- hálózati sávszélesség (network bandwidth)

Ebben a pontban bemutatjuk, hogy az előzőekben a `http://index.hu` helyre elvégzett HTTP GET tesztünket milyen módon lehet feldolgozni. Futtassuk le a Grinder analyzer-t a következő paranccsal:

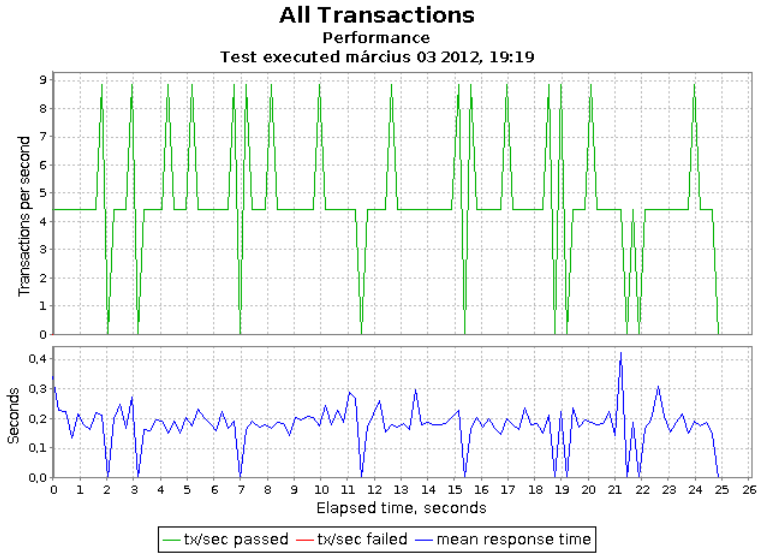
```
jython ./analyzer.py ../log/csdev1-0-data.log ../log/➥
    csdev1-0.log
```

Ekkor a 1.7-től a 1.14-ig mutatott ábrákon lévő grafikonok jönnek létre. A parancs paraméterei mindig azok a napló file-ok, amik a tesztelés során keletkeztek. Az analyzer parancs 1 oldalas manuálja itt olvasható: `http://track.sourceforge.net/usingAnalyzer.html`. A grafikonok mellé egy nagyon informatív összefoglaló táblázat is keletkezik (1.15. ábra). Az értékek jelentése egyértelmű, de nézzük meg együtt is a számokat!

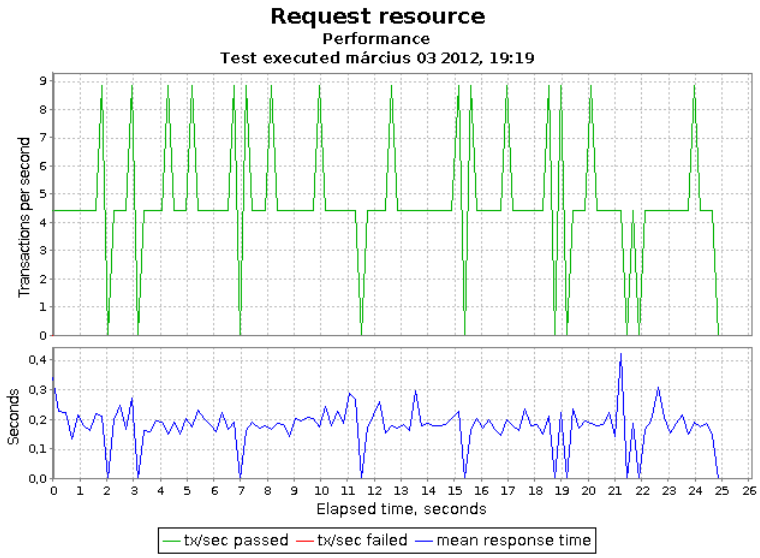
- *Tests Passed*: 117. Ez a sikeresen elvégzett tesztek száma.
- *Tests w/ Errors*: 0. Ennyi teszt futott hibára, esetünkben nagyon jól sikerült a mérés, mert ilyen nem volt.
- *Pass Rate*: 1.00. A sikeres tesztek aránya, esetünkben ez 100%, azaz 1.00.
- *Mean Response Time:* 194,91 ms. Ennyi volt az átlagos válaszidő, amit a 1.9. ábra egy grafikonnal is részletez. Látható, hogy az X tengely a tesztelés lefutási idejét mutatja 0-25 másodpercig. Az Y tengely pedig másodpercben mutatja a válaszidőket, ami ennek az átlagnak megfelelő ~0.2 sec környékén ingadozik.
- *Response time standard dev.*: 46,97 ms. Az értékek szórási intervalluma, ezt szintén a 1.9. ábráról is láthatjuk.
- *Mean Response Length:* 156 145 byte, ami az index induló html lapjának mérete.
- *Bytes per Sec:* 731 665,87. Ekkora sebességgel értük el a hálózatot a teszt alatt, aminek a részleteit a 1.10. ábra tartalmazza.
- *Mean Time Resolve Host:* 0,76 másodperc. A 1.11. ábra egyszerre mutatja ennek az elemeit és statisztikáját.

- *Mean Time Establish Connection:* A TCP/IP kapcsolat kialakítási ideje 21,32 ms.
- *Mean Time to First Byte:* Az első válaszbyte átlagos megjelenési ideje 46,85 ms.
- A táblázat utolsó 4 oszlopa azt mutatja, hogy mennyi tesztfutás volt 1 s alatt, 1-3 s, 3-10 s között vagy 10 s felett.

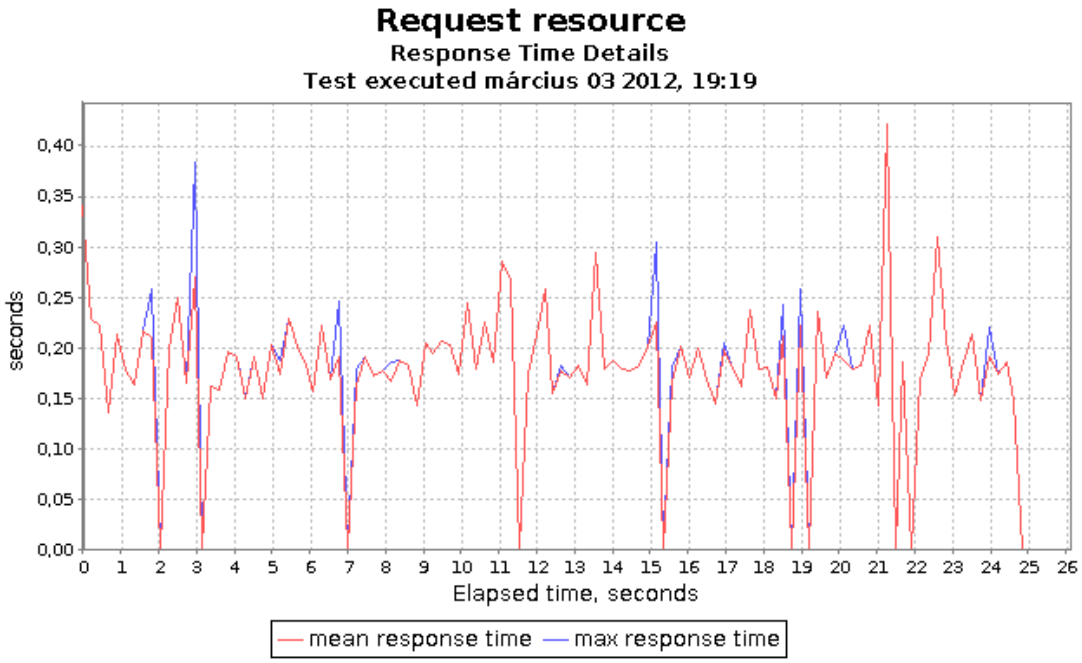
A táblázat *Totals* sora, illetve az „All” jellegű grafikonok akkor lényegesek, amikor több szálon futtatjuk a teszteket.



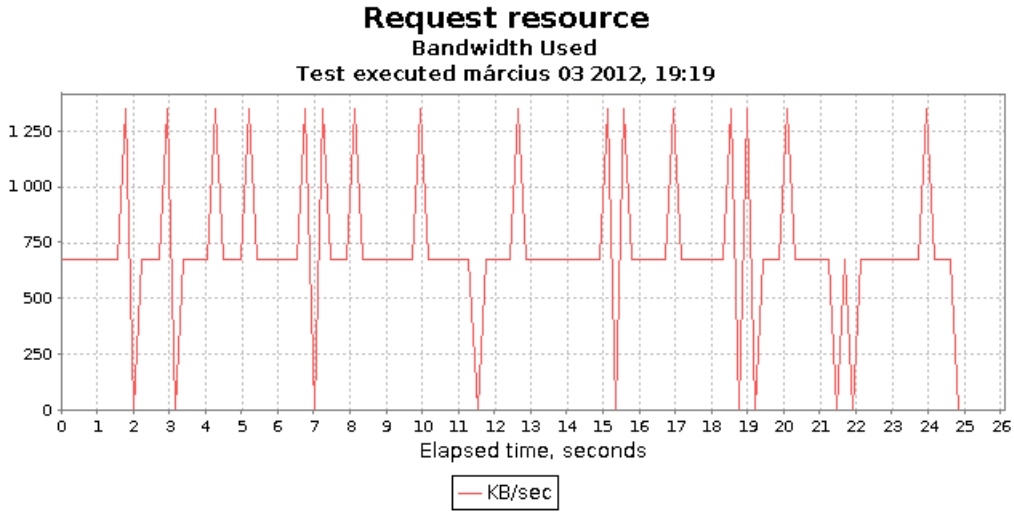
1.7. ábra. All_Transactions.perf.png



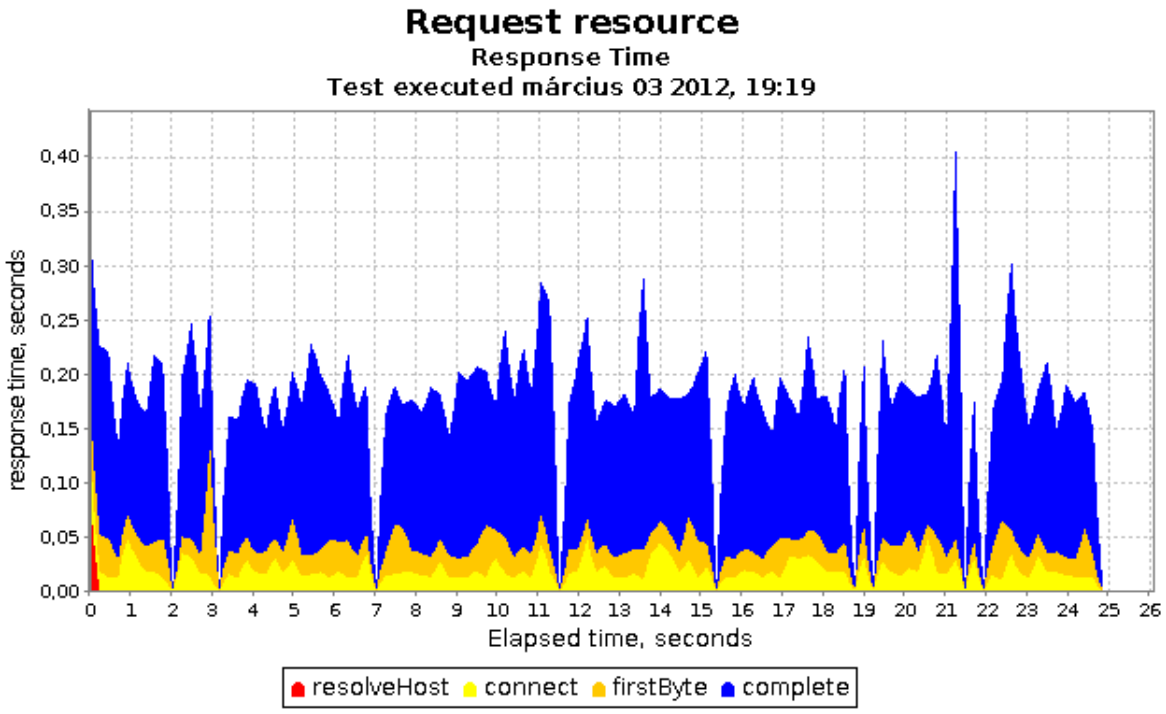
1.8. ábra. Request_resource.perf.png



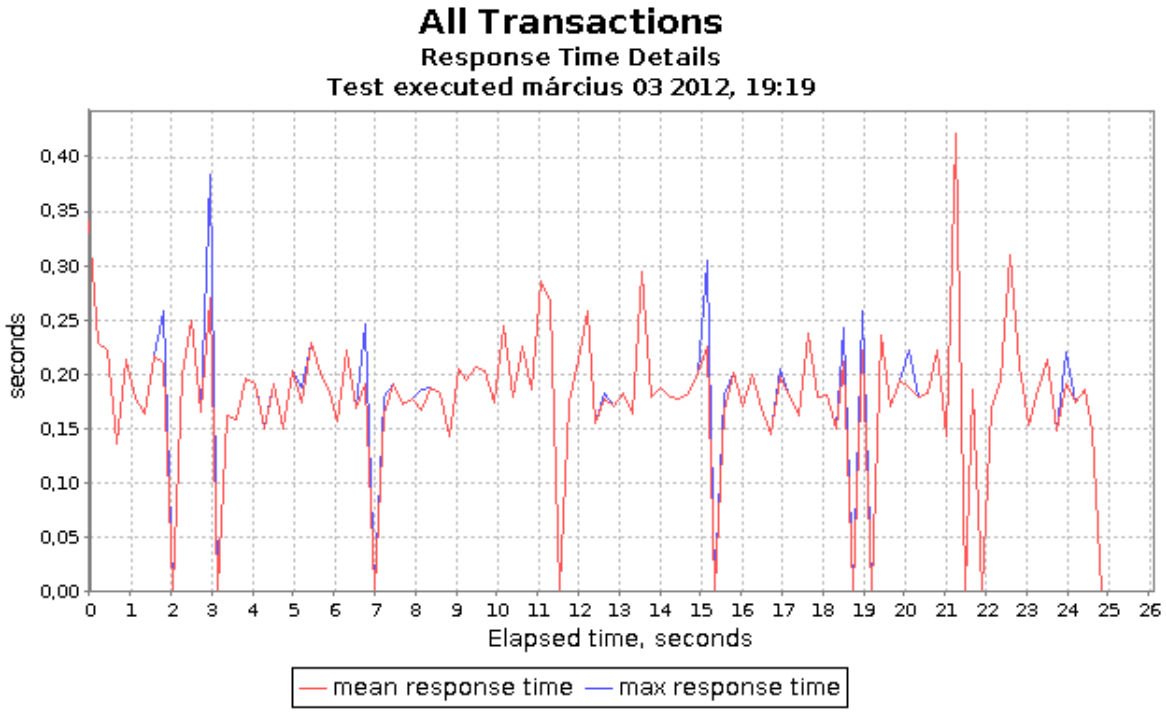
1.9. ábra. Mean Response Time - Request_resource.meanMax_rtime.png



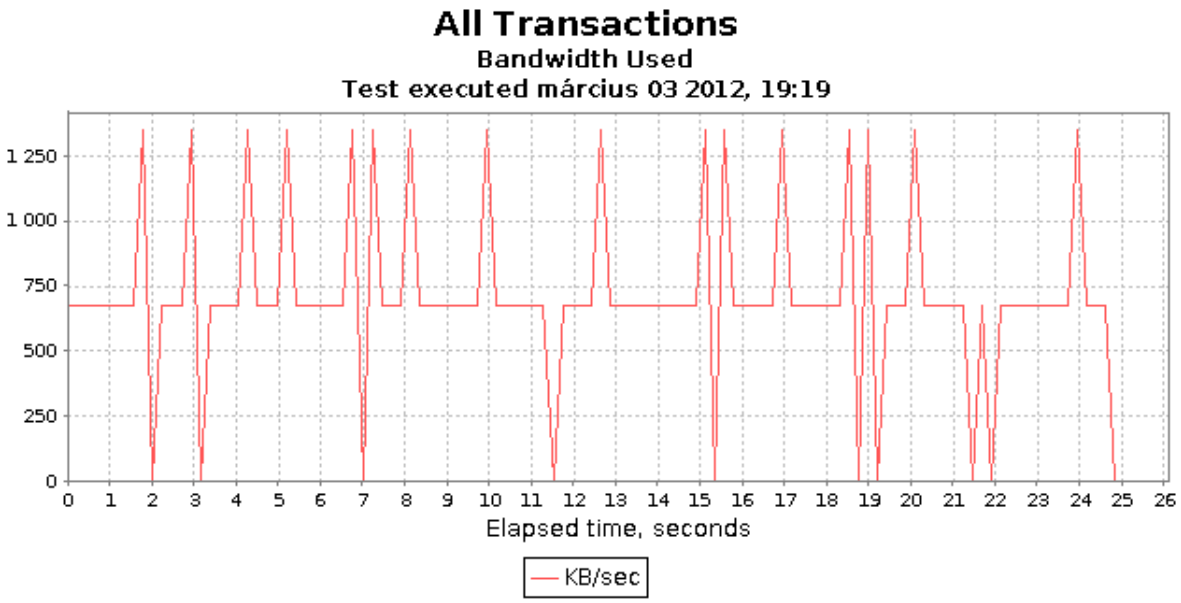
1.10. ábra. Request_resource.bandwidth.png



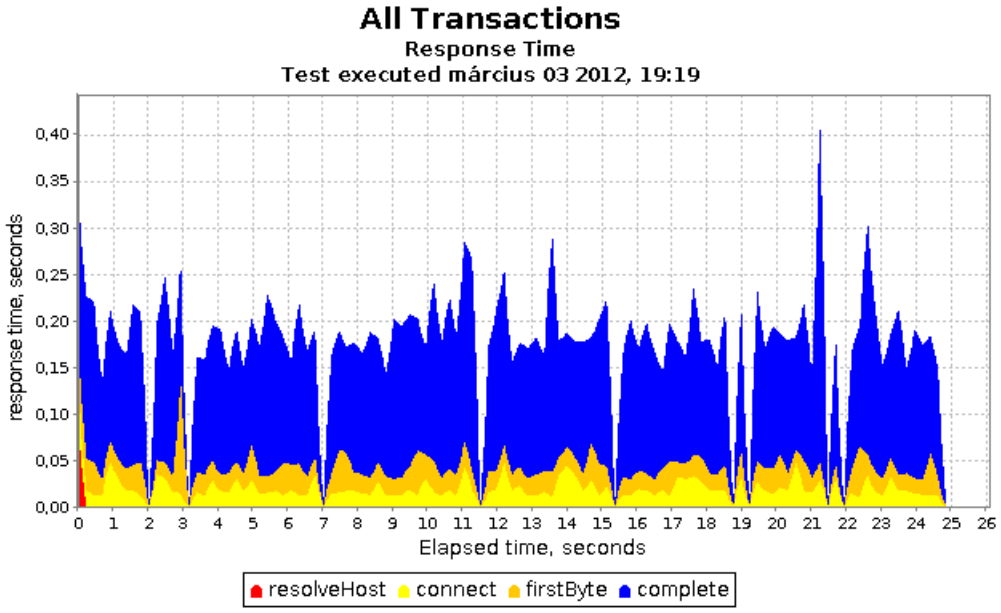
1.11. ábra. Request_resource.rtime.png



1.12. ábra. All_Transactions.meanMax_rtime.png



1.13. ábra. All_Transactions.bandwidth.png



1.14. ábra. All_Transactions.rtime.png

Transaction Name	Tests Passed	Tests w/ Errors	Pass Rate	Mean Response Time	Response time standard dev.	Mean Response Length	Bytes per Sec	Mean Time Resolve Host	Mean Time Establish Connection	Mean Time to First Byte	under 1 sec	1 to 3 sec	3 to 10 sec	over 10 sec
Request resource	117	0	1,000	194,91	46,97	156Â 145	731Â 665,87	0,76	21,32	46,85	1,000	0,000	0,000	0,000
Totals	*117*	*0*	*1,000*	*194,91*	*46,97*	*156Â 145*	*731Â 665,87*	*0,76*	*21,32*	*46,85*	*1,000*	*0,000*	*0,000*	*0,000*

1.15. ábra. Grinder Analyzer összefoglaló táblázat

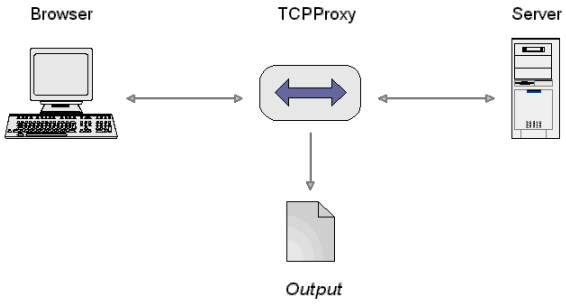
Az elemző eszköznek van egy másik utility parancsa is a *difftool*:

```
jython ./difftool.py [ options ] <file1> <file2>
```

Itt több teszt naplóit lehet megadni, az eszköz pedig készít egy elemző riportot a különbségekről. A használat 1 oldalas manuálja innen olvasható el: `http://track.sourceforge.net/usingDifftool.html`.

A Grinder proxy

A TCP Proxy egy közönséges proxy szerviz és ahogy az 1.16. ábra mutatja, képes a böngésző és a szerver között átfolyatni a TCP adatfolyamot.



1.16. ábra. TCP Proxy

A proxy a következő *startProxy.sh* paranccsal indítható el:

```
// startProxy.sh

#!/bin/bash
. ./setGrinderEnv.sh
java -classpath $CLASSPATH net.grinder.➥
    TCPProxy -console -http > grinder.py
```

Induláskor ez a szöveg kerül ki a képernyőre, azaz a böngésző proxy beállítását a *localhost:8001* portra kell majd beállítani:

```
inyiri@csdev1:/home/tanulas/grinder$ ./startProxy.sh
2012-03-04 08:39:17,296 INFO  tcpproxy: Initialising ➥
    as an HTTP/HTTPS proxy with the parameters:
  Request filters:     HTTPRequestFilter
  Response filters:    HTTPResponseFilter
  Local address:       localhost:8001
2012-03-04 08:39:18,250 INFO  tcpproxy: Engine ➥
    initialised, listening on port 8001
```

Itt most http proxy üzemmódban indítottuk, ahol a proxy-zott output konzol tartalmat röptében python kódra alakítja és esetünkben ezt a *grinder.py* file-ban tárolja. A keletkezett *grinder.py* kód hosszú, ezért az 1-3. programlista ennek csak a részleteit mutatja. Ezzel a módszerrel a böngészős munkát felvehetjük és a már remélhetőleg jól ismert tesztkörnyezettel akárhányszor le is futtathatjuk. Tegyük is meg! A futási eredményt a 1.17 és 1.18 ábrák érzékeltetik. Azt gondoljuk, hogy ez a módszer igazán nagyszerű eszköz a végrehajtható tesztesetek generálására. A proxy képes a http és https tartalmat egyaránt kezelni. A HTTP és SSL beállításokról innen olvashatjuk el a részleteket: `http://grinder.sourceforge.net/g3/tcpproxy.html#ssl`. Sokszor csak a valódi http forgalmat szeretnénk felvenni, ilyenkor a proxy indítása így módosul:

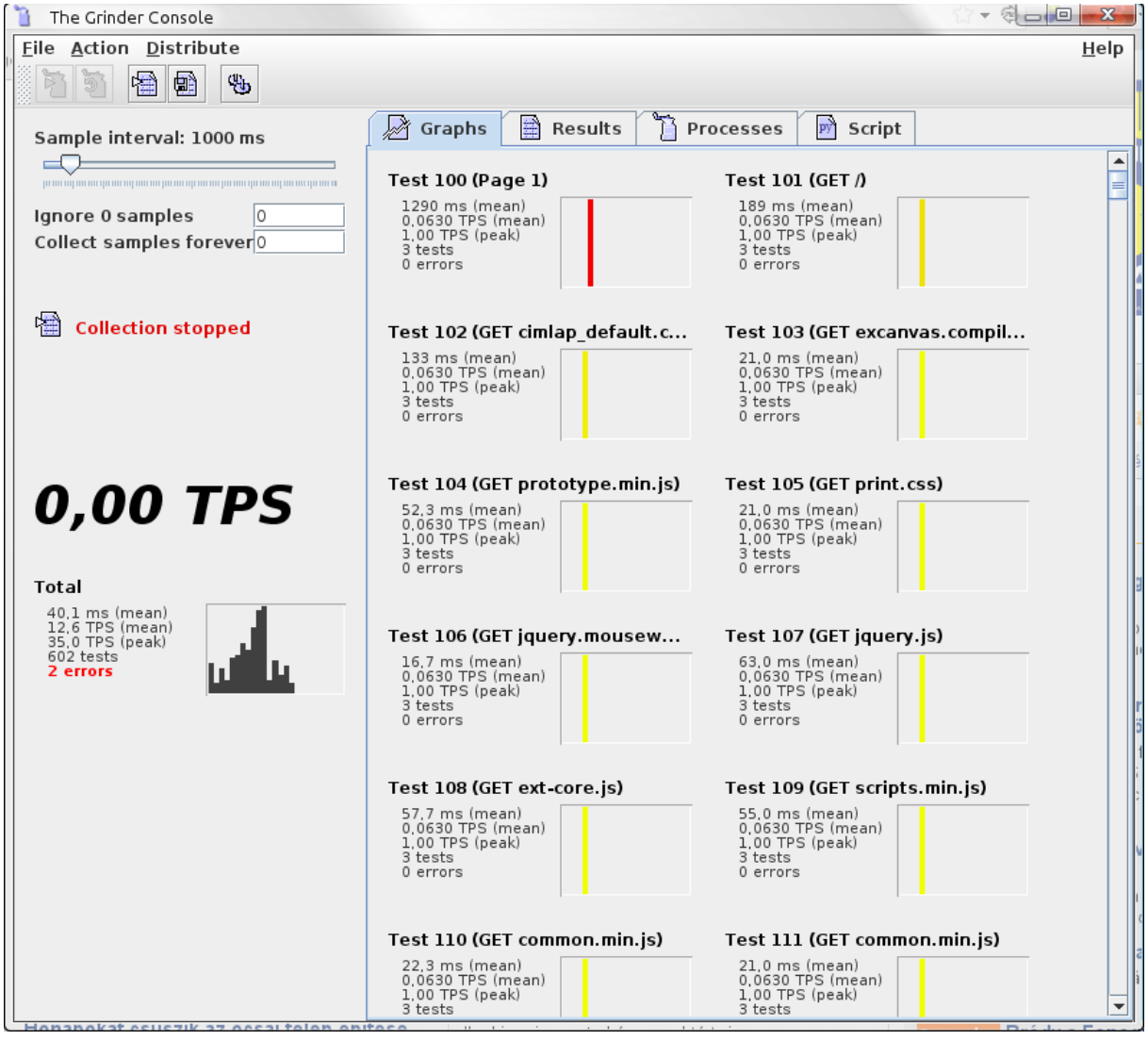
```
#!/bin/bash
. ./setGrinderEnv.sh
java -classpath $CLASSPATH net.grinder.➥
    TCPProxy -console > grinder.log
```

Ez esetben a *grinder.log* egy részlete így néz ki:

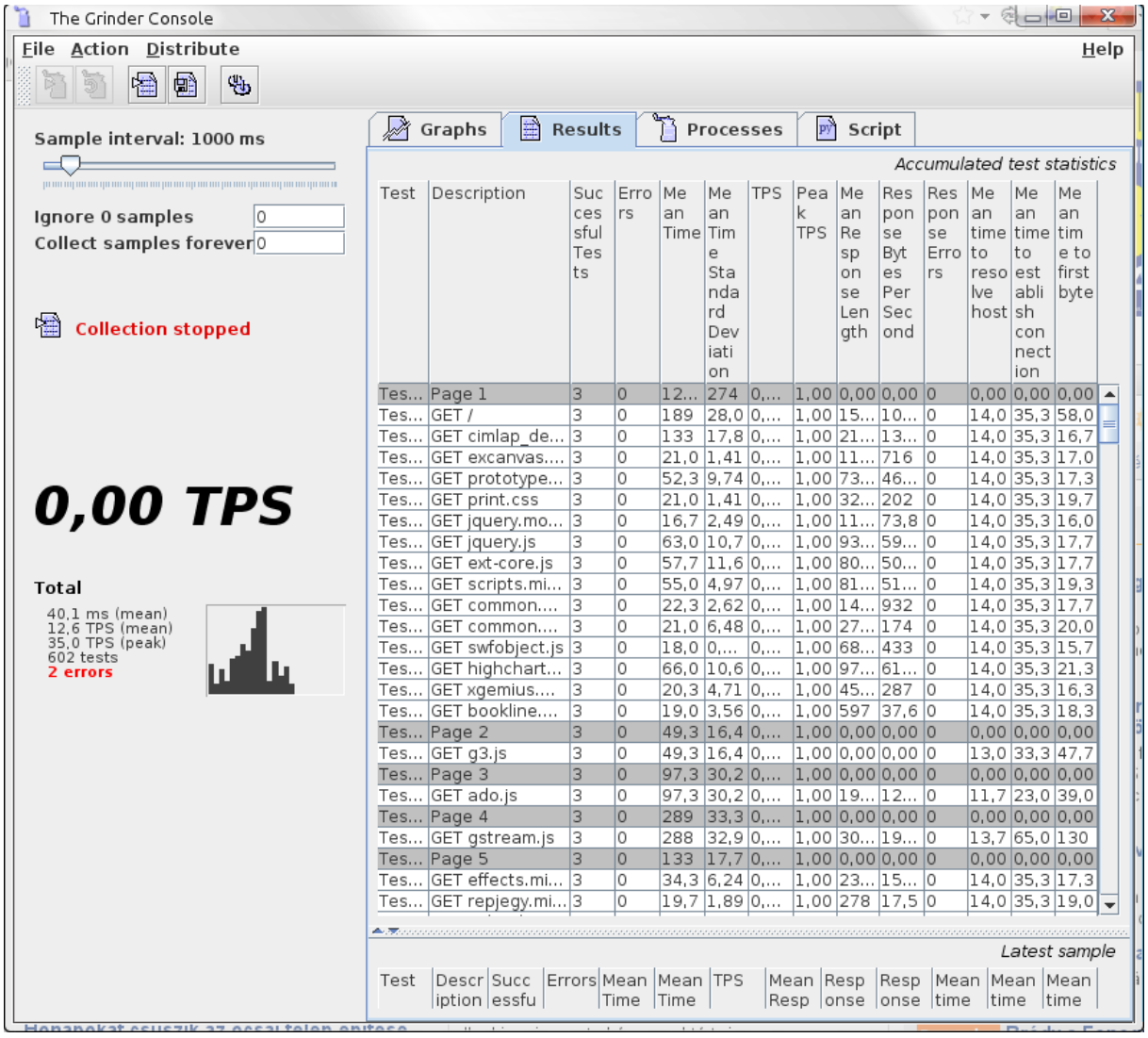
```
------ 127.0.0.1:2263->ads.osdn.com:80 ------
GET /?ad_id=5839&alloc_id=12703&site_id=2&request_id➥
    =8320720&1102173982760 HTTP/1.1
Host: ads.osdn.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.0; ➥
    en-US; rv:1.7.5) Gecko/20041107 Firefox/1.0
Accept: image/png,*/*;q=0.5
Accept-Language: en-gb,en-us;q=0.7,en;q=0.3
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Referer: http://sourceforge.net/projects/grinder
```

```
--- ads.osdn.com:80->127.0.0.1:2263 opened --
------ ads.osdn.com:80->127.0.0.1:2273 ------
HTTP/1.1 200 OK
Date: Sat, 04 Dec 2004 15:26:27 GMT
Server: Apache/1.3.29 (Unix) mod_gzip/1.3.26.1a ➥
     mod_perl/1.29
Pragma: no-cache
Cache-control: private
Connection: close
Transfer-Encoding: chunked
Content-Type: image/gif


------ ads.osdn.com:80->127.0.0.1:2273 ------
80B
GIF89ae[00]) [00D50000C3C3C3FEFDFD]hhhVVVyyy[➥
     F5CCD2D4D4D4CBCBCBD7]'F
```

Amennyiben a gépünk eleve csak egy proxy szerveren keresztül éri el az erőforrást, úgy képesek vagyunk a *-httpproxy* vagy *-httpsproxy* kapcsolót használni a proxy indításakor. Itt a host:port formában kell a külső proxy-t megadni. Az eszköz még a port továbbításra is alkalmas. Aki a TCP Proxy programról még többet szeretne megtudni, itt további olvasnivalót talál: `http://grinder.sourceforge.net/g3/tcpproxy.html`. Az NTLM hitelesítés is használható.



1.17. ábra. Az egyes teszt URL-ek grafikonja



1.18. ábra. Az egyes teszt URL-ek számszerű adatai

```
1  // 1-3. programlista: A http://index.hu helyre generált teszt python script
2
3  # The Grinder 3.7.1
4  # HTTP script recorded by TCPProxy at 2012.03.04. 8:39:17
5
6  from net.grinder.script import Test
7  from net.grinder.script.Grinder import grinder
8  from net.grinder.plugin.http import HTTPPluginControl, HTTPRequest
9  from HTTPClient import NVPair
10 connectionDefaults = HTTPPluginControl.getConnectionDefaults()
11 httpUtilities = HTTPPluginControl.getHTTPUtilities()
12
13 # To use a proxy server, uncomment the next line and set the host and port.
14 # connectionDefaults.setProxyServer("localhost", 8001)
15
16 # These definitions at the top level of the file are evaluated once,
17 # when the worker process is started.
```

```
18
19  connectionDefaults.defaultHeaders = \
20    [ NVPair('Accept-Language', 'hu,en-us;q=0.7,en;q=0.3'),
21      NVPair('Accept-Encoding', 'gzip,␣deflate'),
22      NVPair('User-Agent', 'Mozilla/5.0␣(X11;␣Ubuntu;␣Linux␣i686;␣rv:10.0.2)␣Gecko/20100101␣➥
            Firefox/10.0.2'), ]
23
24  headers0= \
25    [ NVPair('Accept', 'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'), ]
26
27  headers1= \
28    [ NVPair('Accept', 'text/css,*/*;q=0.1'),
29      NVPair('Referer', 'http://index.hu/'), ]
30
31  headers2= \
32    [ NVPair('Accept', '*/*'),
33      NVPair('Referer', 'http://index.hu/'), ]
34  .....
35  .....
36  # Create an HTTPRequest for each request, then replace the
37  # reference to the HTTPRequest with an instrumented version.
38  # You can access the unadorned instance using request101.__target__.
39  request101 = HTTPRequest(url=url0, headers=headers0)
40  request101 = Test(101, 'GET␣/').wrap(request101)
41
42  request102 = HTTPRequest(url=url0, headers=headers1)
43  request102 = Test(102, 'GET␣cimlap_default.css').wrap(request102)
44
45  request103 = HTTPRequest(url=url0, headers=headers2)
46  request103 = Test(103, 'GET␣excanvas.compiled.js').wrap(request103)
47
48  request104 = HTTPRequest(url=url0, headers=headers2)
49  request104 = Test(104, 'GET␣prototype.min.js').wrap(request104)
50  .....
51  .....
52  class TestRunner:
53    """A TestRunner instance is created for each worker thread."""
54
55    # A method for each recorded page.
56    def page1(self):
57      """GET / (requests 101-115)."""
58      result = request101.GET('/')
59      self.token_il = \
60        httpUtilities.valueFromBodyURI('il') # '/'
61      self.token_t = \
62        httpUtilities.valueFromBodyURI('t') # '/24ora/'
63      # 10 different values for token_rov found in response, using the first one.
64      self.token_rov = \
65        httpUtilities.valueFromBodyURI('rov') # '/'
66      # 13 different values for token_a found in response, using the first one.
67      self.token_a = \
68        httpUtilities.valueFromBodyURI('a') # 'http://totalcar.hu/'
69      # 145 different values for token_id found in response, using the first one.
70      self.token_id = \
71        httpUtilities.valueFromBodyURI('id') # 'inxcl'
72  .....
73  .....
```

2. Egységtesztelés a JUnit használatával

Bizonyára sok olvasónk hallott már az egységtesztek fontosságáról és a teszt vezérelt (Test Driven) fejlesztés hatékonyságáról. A módszer alapvető gondolata, hogy a fejlesztés során, azzal párhuzamosan készítjük a teszteseteket is, amivel azonnal ellenőrizni tudjuk a kód működését. Jelentős előny, hogy ezek a tesztesetek a fejlesztés során mindvégig rendelkezésünkre állnak, azokat akármennyiszer újra és újra lefuttathatjuk, amikor változtatunk a kódon. Szép látni, hogy egy-egy nagyobb alkalmazás esetén akár több száz ilyen teszteset is végrehajtódhat minimális energiabefektetés mellett.

Írásunkban az egységtesztelést mutatjuk be az elterjedten használt *JUnit* keretrendszer használata segítségével (webhelye: `http://www.junit.org/`). A *JUnit* egy szabad forráskódú modulteszt rendszer, amely Java programjaink automatikus teszteléséhez nyújt segítséget, így javíthatjuk programjaink minőségét, ugyanakkor hibakeresési időt takaríthatunk meg. A teszt vezérelt fejlesztés (TDD=Test Driven Development) szabályai szerint a kód írásával párhuzamosan fejlesztjük a kódot tesztelő osztályokat is, ezek az egységtesztek. A *JUnit* az egységtesztek karbantartására és csoportos futtatására szolgál. A teszteket gyakran a build folyamat részeként szokták beépíteni. *JUnit* framework fizikailag egy java jar fájlba van csomagolva. A keretrendszer jelenleg 2 elterjedtebb verzióban használatos, amelyekben az osztályok a következő alapcsomag alatt találhatóak:

- *JUnit* 3.8-as vagy korábbi verzióiban a *junit.framework*
- *JUnit* 4-es vagy későbbi verzióiban *org.junit*

JUnit-ot alapul véve más programozási nyelvekre is portolták ezt az egységteszt keretrendszert: C (CUnit), C# (NUnit), C++ (CPPUnit), Delphi (DUnit), Pascal (FPCUnit), PHP (PHPUnit), Python (PyUnit). A *JUnit* keretrendszert 2 világhírű fejlesztő, Kent Beck és Erich Gamma alkotta meg.

A JUnit Hello Word

A JUnit tesztek a *junit.framework.TestCase* osztály leszármazottjai, és a tesztelendő kódot publikus *testMetodus()* nevű metódusokban adjuk meg. A teszt futása során az összes ilyen metódus futtatásra kerül, amelynek háromféle eredménye lehet:

1. Sikeres végrehajtás(pass): ez azt jelenti, hogy a teszt rendben lefutott, és az ellenőrzési feltételek mind teljesültek.
2. Sikertelen végrehajtás (failure): a teszt lefutott, de valamelyik ellenőrzési feltétel nem teljesült.
3. Hiba(error): A teszt futása során valami komolyabb hiba merült fel, például egy Exception dobódott valamelyik tesztmetódus futása során, vagy nem volt tesztmetódus a megadott tesztosztályban.

A lenti példában az első tesztosztályunk neve a *NehanyTest* lesz, amit parancssorból így tudunk végrehajtani:

```
java junit.textui.TestRunner NehanyTest
```

Az osztály kódját a 2-1. programlista tartalmazza.

```
1  // 2-1. programlista: Az első tesztosztályunk (JUnit 3.8)
2
3  package org.cs;
4
5  import junit.framework.TestCase;
6  import junit.framework.TestSuite;
7
8  public class NehanyTest extends TestCase
9  {
10     public NehanyTest()
11     {
12     }
13
14     public NehanyTest(String name)
15     {
16         super(name);
17     }
18
19     public void testHello()
20     {
21         fail("Egyből itt kiakadtam :-)");
22     }
23
24     public void testStringSize()
25     {
26         String s = "Alma a fa alatt";
27         assertTrue("Hosszabb, mint 10: ", s.length() < 10);
28     }
29
30     public static void main(String[] args)
31     {
32         junit.textui.TestRunner.run(new TestSuite(NehanyTest.class));
33     }
34 }
```

A 8. sor mutatja, hogy ez egy *TestCase* osztály, ahol a 14-17 sorok közötti String paraméteres konstruktort is megadtuk. Ez nem kötelező, de amennyiben egy-egy tesztkészletet metódusonként akarunk összeépíteni, úgy egy String paramétert át kell venni, és azt a *super(name)* hívással továbbítani. A használatot a későbbiekben mutatjuk be. A *NehanyTest* class egy teljes tesztosztály, aminek az összes tesztmetódusát lefuttatja a 32. sorban lévő *main()*-beli hívás. Ennek eredményét mutatja a lenti 2-1. Futási kép. Lényeges megjegyezni, hogy a JUnit csak azokat a publikus metódusokat tekinti automatikusan tesztmetódusoknak, amik a *test* előtaggal kezdődnek. Jelen példában 2 ilyen metódusunk van: *testHello()*, *testStringSize()*. Az összes többi metódus csak utility funkciót tölt be az osztályon belül. A példában a *TestRunner* class *run()* metódusát programból használtuk, amihez még fontos megérteni a tesztkészlet (*TestSuite*) fogalmát is. A tesztkészlet azoknak a metódusoknak a halmaza, amiket a *TestRunner* le fog futtatni. Példánkban egy olyan névtelen *TestSuite* objektumot adtunk át a *run()* me-

tódusnak, ami azokat a tesztmetódusokat tartalmazza, amik a *NehanyTest* osztályban *test* szócskával kezdődnek. Megjegyezzük, hogy létezik a *JUnit* csomagon belül grafikus felületű tesztfuttató is, de azt ebben a cikkben nem tárgyaljuk, ugyanis a jelentősége nem túl nagy az automatikus tesztfuttatások során.

Tesztkészletek létrehozása

Talán az előző példából is kitűnt, hogy a JUnit lényeges része, hogy a meglévő *TestCase* osztályokból, hogyan lehet tesztkészletet készíteni, ugyanis az ebben lévő konkrét elemi tesztek végigfuttatása jelenti a unit teszt elvégzését. A 2-2. programlista egy nagyon triviális másik tesztosztályt is mutat, szerepe csak annyi, hogy ne csak 1 db tesztosztályunk legyen, amikor összeépítjük a tesztkészletet.

A 2-3. programlista egy nagyon egyszerű, külön *TesztelendoTestSuite* nevű osztályban megvalósított tesztkészlet összeállítást és annak futtatását mutatja.

```
1  // 2-2. programlista: Egy másik tesztosztály (JUnit 3.8)
2
3  package org.cs;
4
5  import junit.framework.TestCase;
6
7  public class NehanyMasikTest extends TestCase
8  {
9      public void testProba1()
10     {
11         System.out.println("Próba 1");
12     }
13
14     public void testProba2()
15     {
16         System.out.println("Próba 2");
17     }
18 }
```

A 16-21 sorokban a *suite()* statikus metódus egy tesztkészlet, azaz *TestSuite* objektumot ad vissza. A működés itt is az, hogy a 19. sorban átadott *NehanyTest.class* minden test kezdetű metódusa kerüljön be a készletbe. A tesztelés meghívását, azaz a tesztkészlet végrehajtásának módját a *main()* metódusból olvashatjuk ki. A következő példában bemutatjuk azokat a technikákat, amikkel kifinomultabb módon is össze tudunk tesztkészleteket állítani. Az eddigiekhez képest meg fogjuk tanulni, hogy más test suiteökből, vagy akár az egyes metódusokból hogyan építhetünk a céljainkhoz illeszkedő olyan tesztkészletet, aminek egyes elemeit más-más tesztosztályból emeljük ki. Nézzük a 2-4. programlistát! A 17. sortól kezdődő statikus *suite()* metódus most különféle módszerekkel állítja össze a tesztkészlet tesztmetódus halmazát. A 21-22 sorok hagyományos módon létrehoznak 2 új készletet, aminek *suite1* és *suite2* a neve, majd ezeket a 25-26 sorokban hozzáfűzzük a mi felépítendő *suite* objektumunkhoz. A példa 27-28. sorai mutatják, hogy a *NehanyTest* és *NehanyMasikTest* osztályokra mindezt hogyan lehet 1 lépésben is elvégezni, köztes teszt suite létrehozása nélkül.

```
1  // 2-3. programlista: Egy külön tesztkészlet kialakítása
2
3  package org.cs;
4
5  import junit.framework.Test;
6  import junit.framework.TestCase;
7  import junit.framework.TestSuite;
8
9  public class TesztelendoTestSuite extends TestCase
10 {
11     public TesztelendoTestSuite(String testName)
12     {
13         super(testName);
14     }
15
16     public static Test suite()
17     {
18         TestSuite suite = new TestSuite("TesztelendoTestSuite");
19         suite.addTestSuite(NehanyTest.class);
20         return suite;
21     }
22
23     public static void main(String[] args)
24     {
25         junit.textui.TestRunner.run( suite() );
26     }
27 }
```

A 30-39 sorokból azt tanulhatjuk meg, hogy 1 db metódust (esetünkben most ez a *testStringSize()* metódus) hogyan tudunk a tesztkészlethez fűzni. A 42. és 43. sorokból szintén azt leshetjük el, hogy egy-egy metódust mi módon tehetünk a teszthalmazba. Remélhetőleg még emlékszünk arra, hogy a *NehanyTest* class részére implementeltáltunk egy String paraméterű konstruktort is. Nos itt van ennek jelentősége, mert ezen keresztül adható meg, hogy az osztálynak melyik tesztmetódusát szeretnénk befűzni a készlethez.

```
1  // 2-4. programlista: Egy tesztkészlet fejlettebb létrehozása
2
3  package org.cs;
4
5  import junit.framework.Test;
6  import junit.framework.TestCase;
7  import junit.framework.TestSuite;
8
9  public class TesztelendoTestSuite extends TestCase
10 {
11
12     public TesztelendoTestSuite(String testName)
13     {
14         super(testName);
```

```
15      }
16
17      public static Test suite()
18      {
19          TestSuite suite = new TestSuite("TesztelendoTestSuite");
20
21          TestSuite suite1 = new TestSuite(NehanyTest.class);
22          TestSuite suite2 = new TestSuite(NehanyMasikTest.class);
23
24          // Tesztkészletek összefűzése
25          suite.addTest( suite1 );
26          suite.addTest( suite2 );
27          suite.addTestSuite(NehanyTest.class);
28          suite.addTestSuite(NehanyMasikTest.class);
29
30          // 1 darab metódus hozzáadása a tesztkészlethez
31          TestCase testek = new NehanyTest("Ez 1 darab metódus tesztje!")
32          {
33              @Override
34              public void runTest()
35              {
36                  testStringSize();
37              }
38          };
39          suite.addTest(testek);
40
41          // További metódusok hozzáadása a tesztkészlethez - másképpen
42          suite.addTest( new NehanyTest( "testHello" ) );
43          suite.addTest( new NehanyTest( "testStringSize" ) );
44
45          return suite; // visszaadjuk a tesztkészletet
46      }
47
48      public static void main(String[] args)
49      {
50          junit.textui.TestRunner.run( suite() );
51      }
52  }
```

Egy tesztkészlet futtatása

A korábbiakhoz képest itt csak össze szeretnénk foglalni, hogy mit jelent egy automatizált teszt futtatása. Azt már tudjuk, hogy ez mindig egy előzetesen összeállított *TestSuite* objektum futtatását jelenti, amit viszont már több módon is el tudunk készíteni, amire a 2-5. programlista csak egy újabb példa. Futtatásakor a 2-1. Futási kép-nek megfelelő outputot kapjuk, azaz mindkét tesztesetünk sikertelen lett, ahogy azt a 20. sorból ki is olvashatjuk. Kijavítva mindkét tesztmetódust, a teszt is sikeresen lefut (24. sor), amiről csak egy szerény OK (2 tests) informál bennünket.

Eddig csak a tesztek szerkezeti összeállítására és végrehajtására koncentráltunk, így itt az ideje annak is, hogy megnézzük miket kínál nekünk a *JUnit* a tesztmetódusok megalkotásához.

```
1  // 2-5. programlista: A tésztkészlet összeállítása
2
3  package org.cs;
4
5  import junit.framework.Test;
6  import junit.framework.TestCase;
7  import junit.framework.TestSuite;
8
9  public class TesztelendoTestSuite extends TestCase
10 {
11     public TesztelendoTestSuite(String testName)
12     {
13         super(testName);
14     }
15
16     public static Test suite()
17     {
18         TestSuite suite = new TestSuite("TesztelendoTestSuite");
19
20         suite.addTest( new NehanyTest( "testHello" ) );
21         suite.addTest( new NehanyTest( "testStringSize" ) );
22
23         return suite;
24     }
25
26     public static void main(String[] args)
27     {
28         junit.textui.TestRunner.run( suite() );
29     }
30 }
```

```
1  2-1. Futási kép: A teszt futtatási eredménye:
2  ----------------------------------------
3  .F.F
4  Time: 0,004
5  There were 2 failures:
6  1) testHello(org.cs.NehanyTest)junit.framework.AssertionFailedError: Egyből itt ➥
       kiakadtam :-)
7          at org.cs.NehanyTest.testHello(NehanyTest.java:20)
8          at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
9          at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.➥
               java:39)
10         at sun.reflect.DelegatingMethodAccessorImpl.invoke(➥
               DelegatingMethodAccessorImpl.java:25)
11         at org.cs.TesztelendoTestSuite.main(TesztelendoTestSuite.java:53)
12 2) testStringSize(org.cs.NehanyTest)junit.framework.AssertionFailedError: ➥
       Hosszabb, mint 10:
13         at org.cs.NehanyTest.testStringSize(NehanyTest.java:26)
14         at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
15         at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.➥
               java:39)
```

```
16          at sun.reflect.DelegatingMethodAccessorImpl.invoke(➥
                DelegatingMethodAccessorImpl.java:25)
17          at org.cs.TesztelendoTestSuite.main(TesztelendoTestSuite.java:53)
18
19  FAILURES!!!
20  Tests run: 2,  Failures: 2,  Errors: 0
21
22  2-2. Futási kép: A metódusok javítása után:
23  ----------------------------------------------
24  OK (2 tests)
```

Tesztprogramok írása

Miről is szól egy unit teszt? Általánosságban fogalmazva olyan állítások vizsgálatáról, amik a program futása során vagy teljesülnek, vagy nem teljesülnek. Ezek az állítások a program éppen elérhető állapota, azaz a változói fölött megfogalmazott kijelentések (*assert*). A programban ezeket az assert-eket bárhova elhelyezhetjük és vizsgálható velük, hogy odaérve teljesülnek-e azok az elvárások, amik programunk helyes, vagy éppen helytelen működését hivatottak igazolni. Amennyiben egy *assert* hamisnak minősül, úgy a tesztmetódus futása megszakad és a *JUnit* számára ez a teszteset a sikertelen futást jelenti. Az eddig leírtak alapján látható, hogy a tesztmetódusok akkor igazán hatékonyak, ha azokba kellő helyen és kellő számban el tudunk helyezni jól megfogalmazott *assert* állításokat. Az összes, assert-tel kezdődő nevű ellenőrző metódusnak van olyan változata is, amelynek az első paramétere egy hibaüzenet. Ezt a tesztelő környezet írja ki akkor, amikor a teszt megbukik (azaz nem teljesül a feltétel). Nézzük meg tehát, hogy milyen típusú *assert*-eket biztosít a keretrendszer!

fail

Ez az egyetlen olyan eszköz, ami bár nem *assert*, mégis hatékonyan használható. A *fail(String)* „sikertelenné teszi a metódust”, segítségével ellenőrizhetjük, hogy elérünk-e egy bizonyos kódrészt. Ideérve a tesztet azonnal sikertelenné teszi, azaz megbuktatja. Ezt a hívást a nyelv beépített feltételvizsgálataival lehet használni, de az is lehet, hogy csak azt akarjuk megakadályozni, hogy ezen a ponton továbblépjen a teszt futása.

assertTrue és assertFalse

Az *assertTrue(boolean)* ellenőrzi, hogy a paraméter logikai kifejezés igaz-e. Ennek ellentéte az *assertFalse(boolean).*

assertEquals

Az *assertsEquals([String message], Object, Object)* ellenőrzi, hogy az értékek egyenlőek-e (tömbök esetében a referenciát ellenőrzi, nem az értékeket), amihez az objektum *equals()* metódusát használja. Ennek a vizsgálatnak van 2 speciális esete. Az egyik esetben *assertsEquals([String message], int, int)* vizsgálatot végzünk, mint egyszerű skalár egyenlőség. A *assertEquals(*[String message], *double, double, double)* 2 double skalár egyezőségét vizsgálja a megadott tolerancián belül, amit az utolsó double értékben tudunk definiálni.

assertNull

Az *assertNull([message], object)* ellenőrzi, hogy null objektumról van-e szó.

assertNotNull

Az *assertNotNull([message], object)* ellenőrzi, hogy a paraméter nem null objektum-e.

assertSame

Az *assertSame([String], expected, actual)* akkor teljesül, ha a két változó ugyanazon objektumra mutat, azaz a referenciájuk megegyezik.

assertNotSame

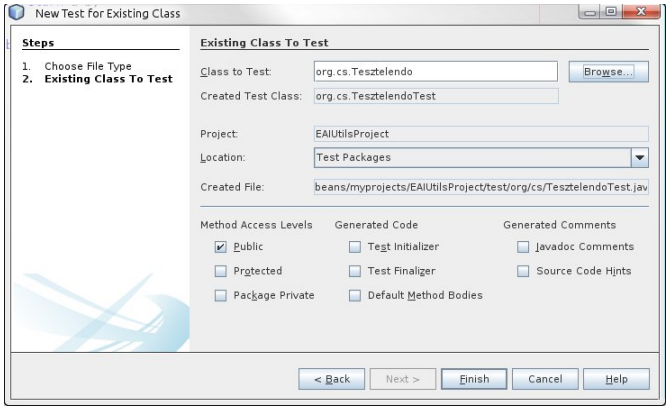
Az *assertNotSame([String], expected, actual)* akkor teljesül, ha a két változó eltérő objektumra mutat, azaz a referenciájuk nem egyezik meg.

A Netbeans JUnit plugin

A továbbiakban megnézzük, hogy a Netbeans környezet hogyan támogatja a *JUnit* tesztek írását. Már itt az elején kiemeljük, hogy egy új Netbeans projekt esetén szinte mindig létrejön a *Source Packages* mellé egy *Test Packages* ág, ugyanis a *JUnit* ide pakolja a teszteléshez kötődő *TestCase* és *TestSuite* osztályokat. Ez kellemes tulajdonság, mert a kód üzleti és tesztelő része mindig elszeparálódik egymástól. A példa kedvéért a 2-5. programlista egy tesztelendő üzleti osztály kódját tartalmazza. A példa természetesen nagyon egyszerű most is, mindössze 2 metódust tartalmaz, de ez a céljainknak tökéletesen megfelelő.

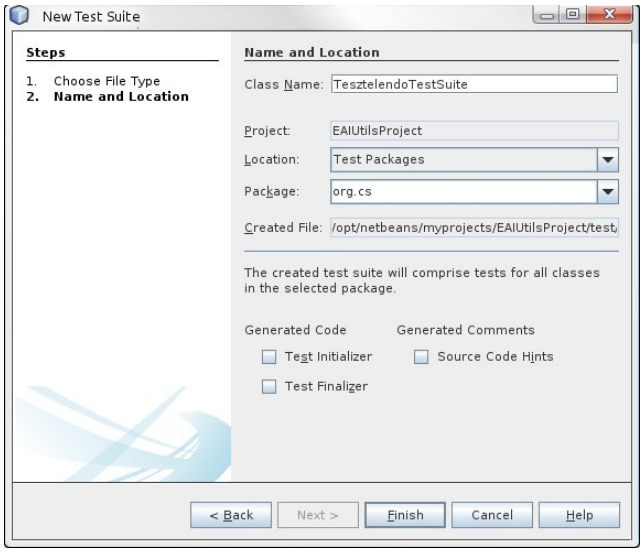
```
1  // 2-5. programlista: Egy tesztelendő osztály
2
3  package org.cs;
4
5  public class Tesztelendo
6  {
7      public int add(int a, int b)
8      {
9          return a+b;
10     }
11
12     public int minimum(int a, int b)
13     {
14         if ( a < b ) return a; else return b;
15     }
16 }
```

A Netbeans támogatás talán leghasznosabb része, hogy minden ilyen osztályra egy *TestCase* osztály generálható, ahogy a 2.1. ábra is mutatja. Mindössze annyi a feladat, hogy jobb click az egérrel és a tesztosztály hozzáadása egy létező osztályra funkciót választjuk ki. Ennek hatására esetünkben létrejön egy ugyanabba a csomagba tartozó üres tesztosztály, azaz az *org.cs.TesztelendoTest*.



2.1. ábra. Egy tesztosztály generálása

Mindegyik tesztelendő metódusra egy tesztmetódust generálódott, ahogy azt a 2-6. programlista mutatja is (a metódusok implementációját persze kézzel írtuk meg). A 2.2. ábra a Netbeans másik *JUnit* támogatását mutatja, segítségével egy tesztkészlet osztályt adhatunk a projektünkhöz. A generálás során megkérdezi, hogy 3.8 vagy 4.x-es *JUnit*-ot szeretnénk-e használni. A 2-5. programlista induló kódját is így állítottuk elő, de persze azt utána már jelentősen átalakítottuk. A Netbeans lényegében az itt leírtakkal támogatja a *JUnit*-ot. Az Eclipse és más hasonló környezetek is hozzávetőleg ilyen támogatást adnak.



2.2. ábra. Egy tesztkészlet generálása

```
1  // 2-6. programlista: A generált teszteset osztály
2
3  package org.cs;
4
5  import junit.framework.TestCase;
6
7  public class TesztelendoTest extends TestCase
8  {
9      public TesztelendoTest(String testName)
10     {
11         super(testName);
12     }
13
14     public void testAdd()
15     {
16         Tesztelendo t = new Tesztelendo();
17         int c = t.add(3, 11);
18         System.out.println("3+11 = " + c);
19         assertEquals(14, 14);
20     }
21
22     public void testMinimum()
23     {
24         Tesztelendo t = new Tesztelendo();
25         int min = t.minimum(12, 8);
26         assertEquals(12, min);
27     }
28 }
```

Annotációk metódusokra - JUnit 4

A JUnit 4 hasznos lehetősége, hogy kihasználva a modernebb Java nyelv lehetőségét, annotációkkal szabályozza a tesztelés körülményeit, illetve pontosabban fogalmazva egy új *TestCase* osztály létrehozását. Tekintsük át emiatt először, hogy melyek ezek a „@”-os utasítások!

@Test

Amikor egy bármilyen nevű publikus metódust *@Test* jelzéssel látunk el, úgy az a *JUnit* számára jelzi, hogy teszt metódusról van szó. A *@Test* annotáció 2 hasznos paraméterezési lehetőséggel is rendelkezik. Az első esetben egy elvárt exception adható meg:

```
import org.junit.*;

public class Junit4ExceptionTest
{
    @Test(expected = ArithmeticException.class)
    public void divisionWithException()
    {
        int i = 1/0;
    }
}
```

Ebben az esetben teszteljük, hogy a metódus egy bizonyos típusú kivételt dob-e, azaz ilyenkor a metódus futása nem számít sikertelennek. A másik paraméterezési lehetőség egy timeout deklarálása a következő formátumban:

```
import org.junit.*;

public class Junit4TimeTest
{
    @Test(timeout = 500)
    public void infinity()
    {
        while (true);
    }
}
```

Sikertelen a teszt, ha a metódus futása tovább tart mint 500 milliszekundum, azaz fél másodperc.

@Before

Az ilyen címkével ellátott metódus minden tesztmetódus hívás előtt végre lesz hajtva, emiatt a tesztelési környezet előkészítésére alkalmazzuk. Ez a lehetőség a JUnit 3 *TestCase* osztályának felülírható *setUp()* metódusát valósítja meg. A cél az, hogy legyen egy olyan inicializálási lehetőség minden tesztmetódus hívás előtt, ami beállítja a változókat, megnyitja a file-okat és/vagy adatbázis kapcsolatokat, stb. Fontos kiemelni, hogy a tesztmetódusok hívási sorrendje általában nem garantált, így nem szabad kihasználni a tesztlépések mellékhatásait, azaz az általuk beállított előző állapotot.

@After

Minden tesztmetódus után végre lesz hajtva, a lefoglalt erőforrások felszabadítására alkalmazzuk. A JUnit 3-ban ez a *tearDown()* felülírásra tervezett metódusnak felel meg.

@BeforeClass

A tesztek futtatása előtt lesz végrehajtva, előkészítő műveletek elvégzésére alkalmazzuk (pl. adatbázis kapcsolat megnyitása).

@AfterClass

Az összes teszt lefuttatása után hívódik meg, erőforrások felszabadítására alkalmazzuk (pl. adatbázis kapcsolat bezárása).

@Ignore

Az így megjelölt tesztmetódus figyelmen kívül hagyódik, azaz nem futtatja le a *TestRunner*. A teszt vezérelt programozás során ezzel jelezzük, hogy a hozzátartozó kód változott és a teszt kódja még nem került megfelelően frissítésre.

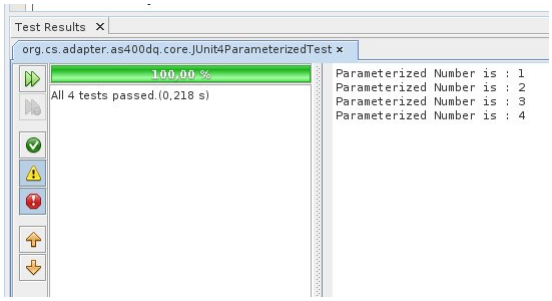
```
import org.junit.*;

public class Junit4IgnoreTest
{
    @Ignore("Not Ready to Run")
    @Test
    public void divisionWithException()
    {
```

```
        System.out.println("Method is not ready
            yet");
    }
}
```

@Parameters

Sok esetben előfordul, hogy ugyanazt a tesztmetódust különféle inputokon akarjuk tesztelni, amire kényelmes megoldást ad a *@Parameters* használata arra a metódusra, amelyik a paraméterek halmazát előállítja. A lenti példában ez a *data()*. A konstruktor feladata ilyenkor, hogy ebből a halmazból mindig vegye a következőt, ahogy ezt a *JUnit4ParameterizedTest* konstruktorának *number* tagjára is látjuk.

```
package org.cs.adapter.as400dq.core;

import java.util.Arrays;
import java.util.Collection;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.
    Parameters;

@RunWith(value = Parameterized.class)
public class JUnit4ParameterizedTest
{
    private int number;

    public JUnit4ParameterizedTest(int number)
    {
        this.number = number;
    }

    @Parameters
    public static Collection<Object[]> data()
    {
        Object[][] data = new Object[][]
        {
            {1}, {2}, {3}, {4}
        };
        return Arrays.asList(data);
    }

    @Test
    public void pushTest()
    {
        System.out.println("Parameterized
            Number is : " + number);
    }
}
```



2.3. ábra. A teszt futtatása Netbeans-ben

A 2.3. ábra mutatja *JUnit4ParameterizedTest* futási képét grafikus tesztfuttató környezetben. Ehhez csak annyit kellett tennünk, hogy jobb egér click és futtat, a többit a Netbeans JUnit plugin elintézi. Amennyiben szöveges módban magunk akarjuk a futtató kódot is megírni, úgy ez a *main()* metódus hozzáadásával a legkönnyebb:

```
public static void main(String[] args)
{
    Result r = org.junit.runner.JUnitCore.
        runClasses( JUnit4ParameterizedTest.
        class );

    for (int i=0; i<r.getFailureCount();i++)
    {
        System.out.println(r.getFailures().get(
            i));
    }
}
```

@RunWith és @Suite

Ezzel a 2 annotációval létező tesztosztályokból (esetünkben ez most a következő kettő: *JunitTest1.class* és *JunitTest2.class*) építhetünk össze egy futtatható tesztkészletet a következő módon:

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses(
{
      JunitTest1.class,
      JunitTest2.class
})
public class Junit4Test
{
}
```

Egy JUnit 4 tesztosztály

Az előzőekben tanult lehetőségeket a 2-7. programlista segítségével szeretnénk röviden bemutatni. A 12. sortól kezdődő *checkConfigFile()* tesztmetódus már jobban hasonlít egy éles tesztesethez, láthatjuk, hogy az *assert*-ek segítségével a futás során különféle helyeken végezhetőek státusz ellenőrzések. Szeretnénk kiemelni, hogy a 41. sorban lévő *anException()* metódus kivételt dob, de ez nem lesz hibás tesztfutás, hiszen a *@Test* annotációnál ezt jeleztük.

```
1  // 2-7. programlista: Egy JUnit 4 tesztosztály
2
3  package org.cs.adapter.as400dq.core;
4
5  import java.io.File;
6  import static org.junit.Assert.*;
7  import org.junit.Test;
8
9  public class BasicTests
10 {
11     @Test
12     public void checkConfigFile() throws Exception
13     {
14         // Is it correct name?
15         String name = "as400dq-adapter-config.xml";
16
17         // Is it correct file name type?
18         if ( !name.endsWith("xml") )
19         {
20             fail("Hibás a konfigurációs file kiterjesztése");
21         }
22
23         // Correct name?
24         assertEquals("Correct config file name?", name, "as400dq-adapter-config.➥
               xml");
25
26         // Exist config file on path?
27         File file = new File("/bea/molconfig/AdminServer/" + name);
28         assertFalse("Check config XML", file.exists());
29     }
30
31     @Test
32     public void runToFail() throws Exception
33     {
34         System.out.println("Indul...");
35         System.out.println("Ideért");
36         if ( 2 == 2 ) {    fail("Bummm!");    }
37         System.out.println("Nem jött ide");
38     }
39
40     @Test(expected = Exception.class)
41     public void anException() throws Exception
42     {
```

```
43          throw new Exception("Dobtam egy kivételt!");
44      }
45  } // end class
```

A fenti *BasicTests* osztályra épülve a 2-8. programlista bemutatja a tesztkészlet egy lehetséges létrehozását és futtatását. A példa érdekessége, hogy megmutatja azt is, hogyan tudunk készíteni JUnit 3 típusú tesztkészletet JUnit 4 tesztosztály használatával, amihez a *JUnit4TestAdapter* osztályt kell alkalmazni.

```
1   // 2-8. programlista: Egy tesztkészlet, ami tartalmazza a BasicTests osztályt
2
3   package org.cs.adapter.as400dq.core;
4
5   import junit.framework.JUnit4TestAdapter;
6   import junit.framework.TestSuite;
7   import org.junit.AfterClass;
8   import org.junit.BeforeClass;
9   import org.junit.Test;
10  import org.junit.runner.RunWith;
11  import org.junit.runners.Suite;
12
13  @RunWith(Suite.class)
14  @Suite.SuiteClasses(
15  {
16      org.cs.adapter.as400dq.core.AS400DataQueueAdaptersTest.class,
17      org.cs.adapter.as400dq.core.BasicTests.class
18  })
19  public class AS400DataQueueAdaptersTestSuite
20  {
21      @BeforeClass
22      public static void setUpClass() throws Exception
23      {
24      }
25
26      @AfterClass
27      public static void tearDownClass() throws Exception
28      {
29      }
30
31      public static junit.framework.Test suiteAS400DataQueueAdaptersTest()
32      {
33          return new JUnit4TestAdapter( AS400DataQueueAdaptersTest.class );
34      }
35
36      public static junit.framework.Test suiteBasicTests()
37      {
38          return new JUnit4TestAdapter( BasicTests.class );
39      }
40
41      public static void main(String[] args)
```

```
42      {
43          System.out.println("Start tests...");
44          junit.textui.TestRunner.run( suiteBasicTests() );
45  //          junit.textui.TestRunner.run( suiteAS400DataQueueAdaptersTest() );
46          System.out.println("End tests...");
47      }
48  }
```

A példában látott *JUnit4TestAdapter* osztályon keresztül használt megoldás lehetővé teszi a korábban megismert összetett JUnit 3 típusú készletek összeépítését is, ahogy a következő 3 sor demonstrálja:

```
junit.framework.Test suite = suiteBasicTests();
TestSuite s1 = new TestSuite();
s1.addTest(suite);
```

Záró gondolatok

Minőségbiztosítás

Egy alapos tesztrendszer mindig pontos képet ad arról, milyen állapotban van a programunk, emiatt segít annak a tervezésében is, hogy mikor adhatunk ki új verziót. Bátrabban állhatunk neki nagyobb kódátszervezési feladatoknak is (refaktorálás), hiszen ha a végén a teszt lefut, akkor biztosak lehetünk benne, hogy nem rontottunk el semmit. Ugyanakkor a tesztek mintegy minőségi tanúsítványt is jelentenek. A tesztek írása hibakereséssel és javítással töltött időt takarít meg. Amikor egy hibajelentés érkezik a programról, érdemes az új hibát is bevenni a tesztrendszerbe, így biztosíthatjuk, hogy ugyanaz a bug remélhetőleg nem kerül ismét vissza a programba.

Teszt vezérelt programozás

A tesztek írását érdemes párhuzamosan végezni a programok írásával. A tesztjeinket mindig tartsuk 100%-ig szinkronban a tesztelendő kóddal, ugyanis ha a tesztek nem az aktuálisak, akkor nem sokat érnek. Érdemes rendszeres időközönként lefuttatni a teszteket. Az Extrém Programozási módszertan (extreme programmming — XP) nagy hangsúlyt fektet a programok tesztelésére. Eszerint a teszteket a programok előtt kell megírni, ezzel mintegy specifikálva a feladatot.

A munka hatékonyságnövelése

A tesztrendszer rögtön környezetet teremt a tesztelendő kód számára, így egyből a funkcionalitásra koncentrálhatunk. A JUnit tesztek automatikusak, azaz nem kell kézzel összehasonlítani a kapott és a várt eredményeket. A tesztek futtatása után azonnal megkapjuk a választ, hogy a programunk jó-e vagy sem. A JUnit tesztek csoportosíthatósága miatt könnyű egybefogni egy nagyobb programrendszer összes tesztjét, és azokat egyszerre futtatni. Ugyanakkor a teszthierarchia egyes részei egyenként is futtathatók. Sokszor a teszteket ugyanolyan csomaghierarchiában tárolják mint a tesztelendő osztályokat. Ez abból a szempontból hasznos, hogy így az osztályok csomagszintű láthatósággal rendelkező tagjai is közvetlenül tesztelhetők.

Speciális tesztelési feladatok

Sajnos sok hasznos tulajdonsága ellenére a JUnit rendszer nem minden tesztelési feladatra alkalmas. A JUnit honlapján számos kiterjesztése is elérhető. Ilyen például a GUI készítésekhez használható Abbot (`http://abbot.sourceforge.net/doc/overview.shtml`). A JUnit további kiterjesztései lehetőséget adnak párhuzamos programok, program hatékonyság (sebesség), JEE programok tesztelésére is.

3. Grinder - Java Enterprise programok tesztelése

Az első írásunk a Grindert átfogóan bemutatta, ezért itt azt a célt tűzzük ki, hogy az eszköz általános használati módját is bemutassuk. Mindezt gyakorlati példákon keresztül, elsősorban a *Java Enterprise Edition* (JEE) lehetőségeire fókuszálva. Véleményünk szerint ez egy izgalmas lehetőség, hogy a teszteljük adatbázis, message queue és más JEE komponensek terhelhetőségét és funkcionális működését.

A Python tesztprogramok

A Grinder tesztprogramok *python* nyelven készülnek és azt annak egy Java implementációja, a *jython* futtatja. Ezzel olyan python programokat készíthetünk, ahol a teljes Java apparátust is megmozgathatjuk, így a JEE megoldások tesztelésének ez egy remek eszköze. A python nyelvet nem kell teljes mélységben érteni, de azért bizonyos nyelvi ismeretek elengedhetetlenek egy-egy tesztprogram megírásához. Természetesen itt nincs hely a nyelv bemutatására, de a példákat azért igyekszünk úgy ismertetni, hogy azokat az is megértse, aki ezen a nyelven nem szerzett még tapasztalatokat, de a Java, C++ vagy C# nyelvek valamelyikét ismeri. A tesztprogramokban vannak közös megoldások, amit itt előre összefoglalunk, hogy a konkrét példáknál már ne kelljen azokat elmagyarázni. A következőkben ezeket ismertetjük röviden, itt sok helyen felhasználtuk az ELTE következő oktatási anyagát: `http://nyelvek.inf.elte.hu/leirasok/Python/index.php`.

Függvények

A függvény definiálásának szintaxisa:

```
def <fvnév> ( <paramlist> ) :
        <megvalósítás>
```

A függvény törzse a következő sorban beljebb kezdve kell, hogy legyen. A törzs opcionálisan kezdődhet egy string literállal, ami a függvény dokumentáció stringje (docstring). A return utasítás teszi lehetővé, hogy a függvény értéket adjon vissza. A függvény által visszaadott értékek egymásnak paraméterként is átadhatók. Csak egyetlen érték vagy objektum visszaadása engedélyezett, de tuple vagy lista adattípust használva több értéket is vissza tudunk adni. Az eljárás egy olyan függvény, amely a speciális *NONE* értéket adja vissza. Példa egy függvényre, aminek *page* a neve:

```
def page():
    request.GET('/console')
    request.GET('/console/login/LoginForm.jsp')
    request.GET('/console/login/bea_logo.gif')
```

Ez pedig egy másik példa (a lenti *jmssender.py* példából kiemelve), ahol 2 db paraméter is van, illetve a *message* egy visszaadott érték:

```
def createBytesMessage(session, size):
    bytes = zeros(size, 'b')
    random.nextBytes(bytes)
    message = session.createBytesMessage()
    message.writeBytes(bytes)
    return message
```

A függvény hívása így néz ki, de a ';' karakter nem kötelező.

```
message = createBytesMessage(session, 1024);
```

A függvényekre alias neveket is megadhatunk, például:

```
csinaldUzenet = createBytesMessage;
message = csinaldUzenet(session, 1024);
```

Lambda kifejezések

A funkcionális programozás hívőinek megmelengetheti a szívét az egyszerű lambda-függvények kezelése. A lambda kulcsszó használatával kisméretű anonymous függvények definiálhatók, amire egy példa:

```
fv = lambda a, b: a+b
```

Ez a két paraméter összegét adja vissza. A Lambda-formák bárhol használhatóak, ahol függvényobjektumokra van szükség. Szintaktikusan egy kifejezésre korlátozottak, szemantikusan pedig csak "szintaktikus cukra" a normál függvénydefiníciónak. Például az *fv(3, 4)* hívás eredménye 7 lesz.

Modulok és importok

A Python lehetővé teszi, hogy nagyobb terjedelmű scripteket, programcsomagokat a belső struktúrájának megfelelően tagoljunk és ezt a programszövegben is jelezzük. Erre szolgálnak a modulok, amelyek logikailag összetartozó programrészeket fognak egybe. A modul neve az azt tartalmazó file neve. A modulok kiterjesztése *.py*. Egy modul neve a modulon belül a *__name__* modul-globális szimbólumon keresztül férhető hozzá. Modulokat más scriptekből az *import* vagy *from module import ** utasításokkal importálhatunk. Az első forma a teljes modult, a második a modul bizonyos függvényeit importálja. A '*' helyett vesszővel elválasztott felsorolás is állhat. A második formában a '*' használatánál az '_'-fal kezdődő függvények nem importálódnak. A modulok állományrendszerbeli keresése a *PYTHONPATH* környezeti változóban felsorolt könyvtárak sorrendjében történik. Hasonlóan a C++-beli *namespace*-ekhez, a különböző nyelvi elemeknek pythonban is létezik struktúrája. A struktúra különböző szintjeit '.'-tal választjuk el egymástól. A névterek általában láthatósági tartományokat is jelölnek. A névtér tulajdonképpen név → objektum leképezések halmaza, jelenleg így is vannak implementálva (szótárakkal). Az általános felhasználásra szánt programcsomagokat csomagokba szervezzük, melyek tetszőleges mélységű könyvtár-struktúrával rendelkezhetnek, amit a Java-hoz hasonlóan a file rendszer szerkezetével reprezentálunk. Az *__init__.py* nevű speciális állományokra minden csomagban szükség van, hogy a Python csomagként kezelje a könyvtárakat. Ez lehet üres, vagy tartalmazhat inicializációs kódot, ami a csomag importálásánál fut le.

Osztályok

A python osztályok szerepe megegyezik a más objektumorientált nyelveknél megszokottakkal. Lehetőség van többszörös öröklődésre, a származtatott osztály átdefiniálhatja az ősosztálya metódusait és egy metódus hívhatja az ősosztály metódusát ugyanazon a néven. Az objektumok tartalmazhatnak privát adatokat. A dupla '_'-sal kezdődő tagokat a parser a szintaktikai előfordulástól függetlenül '*_classname__member*'-re cseréli ki, ahol *classname* az aktuális osztály neve. Hasonlóan a Javahoz, minden tagfüggvény virtuális. Az osztályok maguk is objektumok, ugyanis a Pythonban minden adattípus objektum. Ugyanakkor minden objektumnak van osztálya. Az osztályobjektumokon kétfajta műveletet végezhetünk. Az egyik művelet az attribútumhivatkozások létrehozása, a másik pedig az osztály egy példányának létrehozása. A beépített típusokat nem bővítheti a felhasználó, azaz nem örökölhet tőlük.

Példa egy osztály létrehozására és példányosítására:

```
class Complex:
        def __init__(self, realpart, imagpart):
                self.r = realpart
                self.i = imagpart

x = Complex(3.0,-4.5)
```

Amennyiben létezik egy *__init__()* konstruktor metódusa az osztálynak, akkor példányosításkor az objektum létrehozása után meghívódik, átadva a példányosításkor esetleg megadott paramétereket. A python osztályok rendelkeznek még néhány előre elkészített metódussal, de a Grinder példákhoz most csak a *__call__(self, *argumentumok)* megértése lényeges, ami meghívódik, amikor egy példányt függvényként hívunk meg. Tegyük fel, hogy a fenti *Complex*

osztálynak van egy *__call__(arguments)* metódusa. A példában az x változóra ezt így hívhatnánk meg: *x.__call__(arguments)*, de ezt megtehetjük így is: *x(arguments)*. Talán még a következő 2 metódust is érdemes megértenünk:

- *isinstance(ob1, ob2)*, ahol az első argumentum egy példányobjektum, a második pedig egy osztályobjektum vagy típusobjektum. Azt vizsgálja, hogy az ob1 az ob2 osztály egy példánya-e, vagy, hogy az ob1 típusa megegyezik-e az ob2 típusával.
- *issubclass(osztály1, osztály2)*: megvizsgálja, hogy az osztály1 az osztály2 alosztálya-e. Egy osztály a saját alosztályának tekinthető.

Listák, Sorok és Szótárak

A python lista (*list*) és sor (*tuple*) ugyanolyan lista adatszerkezet, a különbség az közöttük, hogy a *tuple* egyszer felveszi az értékét és már nem változtatható meg, azaz például nem ragasztható hozzá új elem. Mindkettőnek van literálja, azaz konstans alakja. A lista konstansot [...], míg a tuple-t (...) jelek között kell megadnunk. Mindkét típus képes egy olyan másolatot adni magáról, ami a másik típusba tartozik. A tuple azért jelenik meg külön elemként a nyelvben, mert kihasználva annak konstans jellegét, hatékonyabban valósítható meg, így ezekben a szituációkban nem érdemes listát használni. A szótárak (*dictionary* vagy *map*) az ismert kulcs, érték párok halmazának megvalósítása, ahol a kulcs alapján asszociatív (direkt) elérés valósul meg. Ennek a konstansát {...} jelek között adhatjuk meg. Mindhárom típusszerkezet eltérő típusú értékeket tartalmazhat. Az eddigiekre egy rövid szemléltető példa a következő:

```
# coding=utf-8

listaObj = [ 12, "Alma", 34 ]
print listaObj

tupleObj = ( 12, "Alma", 34 )
print tupleObj

dictObj = { "k1":7, "k2":15, "k3":"aaaaa" }
print dictObj

Futási eredmény:
[12, 'Alma', 34]
(12, 'Alma', 34)
{'k3': 'aaaaa', 'k2': 15, 'k1': 7}
```

Alias név egy objektumra

Az alias egy másik név egy osztályra vagy valamilyen adatra. A legegyszerűbb formája az ugyanarra az objektumra való hivatkozás:

```
var = 'Valami'
var2 = var
```

Ez más nyelveken is így van, de a python lehetővé teszi, hogy egy class nevére is aliast-t tegyünk, hiszen ez is egy objektum:

```
class MyReallyBigClassNameWhichIHateToType:
    def __init__(self):
        <blah>
    [...]

És erre az alias név:
ShortName = MyReallyBigClassNameWhichIHateToType
```

Megjegyezzük az első cikkben már megismert metódusra, a *grinder.logger.info(...)* függvényre így tettünk alias nevet:

```
log = grinder.logger.info
```

A *TestRunner* class

Amikor egy Grinder worker process elindul, akkor az éppen megadott teszt script lefut. Minden ilyen script kötelezően tartalmaz egy *TestRunner* osztályt, amiből a Grinder motor létrehoz az összes worker futási szál számára egy-egy példányt, ezek természetesen a szálra jellemző specifikus információkat is tárolhatnak. Ezek a példányváltozók *callable* típusúak, mert az osztálynak kötelezően tartalmaznia kell a már megismert *__call__(...)* metódust. A teszt scriptek a szolgáltatásokat a *grinder* objektumon keresztül érhetik el.

A *Test* class

A *net.grinder.script.Test* class reprezentál egy konkrét tesztesetet, ami a Grinder statisztikában is megjelenik a tesztfutás során. Van egy egyedi azonosítója és rövid szöveges leírása, amely 2 paramétert mindig a konstruktornak kell átadni. Egy *Test* instance használatánál a *wrap(java.lang.Object target)* metódus a legfontosabb, ugyanis ez egy proxy objektumot hoz létre, aminek használati felülete a paraméterként átadott objektuméval egyezik meg. A hívást a Grinder fogja delegálni ezen a proxy-n keresztül a célobjektum felé. Ez a metódus sokszor meghívható. A *wrap()* metódus egy python algoritmus becsomagolása tesztelési célból.

Egy HTTP tesztprogram

Az első JEE tesztprogram az Oracle Weblogic 11g webes konzol login képernyőjének elérését méri. A Weblogic a *localhost:7001* porton hallgatózik. A *grinder.properties* file-ban a *runs* értéke 0, azaz addig hajtódik majd végre a 3-1. programlista, ameddig a konzolról le nem állítjuk azt. A 14. sorban létrehozott *request* objektum http kéréseket tud adni az előbb megnevezett portra. A 16-19 sorok között definiáljuk a *page1()* metódust, ami 3 http request-et ad ki. A 21. sorban hozzuk létre a tesztprogram egyetlen teszt esetét, aminek az azonosítója 1, rövid neve pedig *First page* lett. Itt a *page1Test* változó már ennek a teszt esetnek a becsomagolt változata, ahol a csomag a *page1()* metódust hajtja végre, mert konkrét tesztet. Korábban már tanultuk, hogy a Grinder úgy hívja meg a tesztet, hogy a *TestRunner*-t futtatja, ennek pedig most csak ez az egyetlen *page1Test()* hívás a feladata. Az említett runs=0 miatt ezt most annyiszor hívja, amennyiszer tudja. A futás naplójának egy részletét mutatja a 3-1. Naplórészlet. Látható a végén, hogy 34786 ms alatt 7320 *TestRunner* futás sikerült. A fontosabb statisztikai értékek a következők, amiket a napló legvégéből lehet kiolvasni:

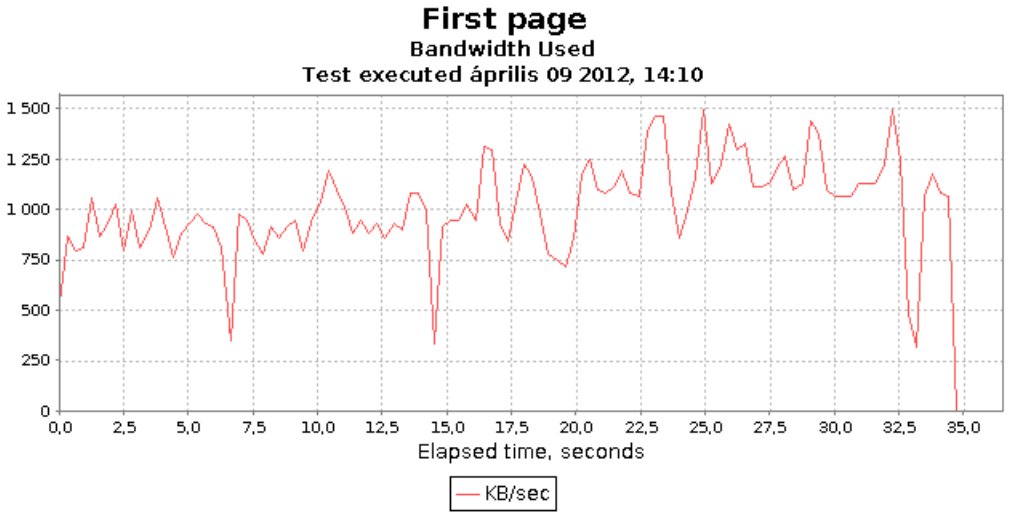
- Mean Test: 3,96 ms
- Standard Deviation: 6,26 ms
- TPS: 210,43
- Mean response length: 4934,00 byte
- Response bytes per second: 1038259,07

A 3.1. és 3.2. ábrák a Grinder elemző által generált grafikonok, ahol fel lett tüntetve a teszt *First page* neve. A grafikonok az idő függvényében mutatják a fenti számok részleteinek alakulását.

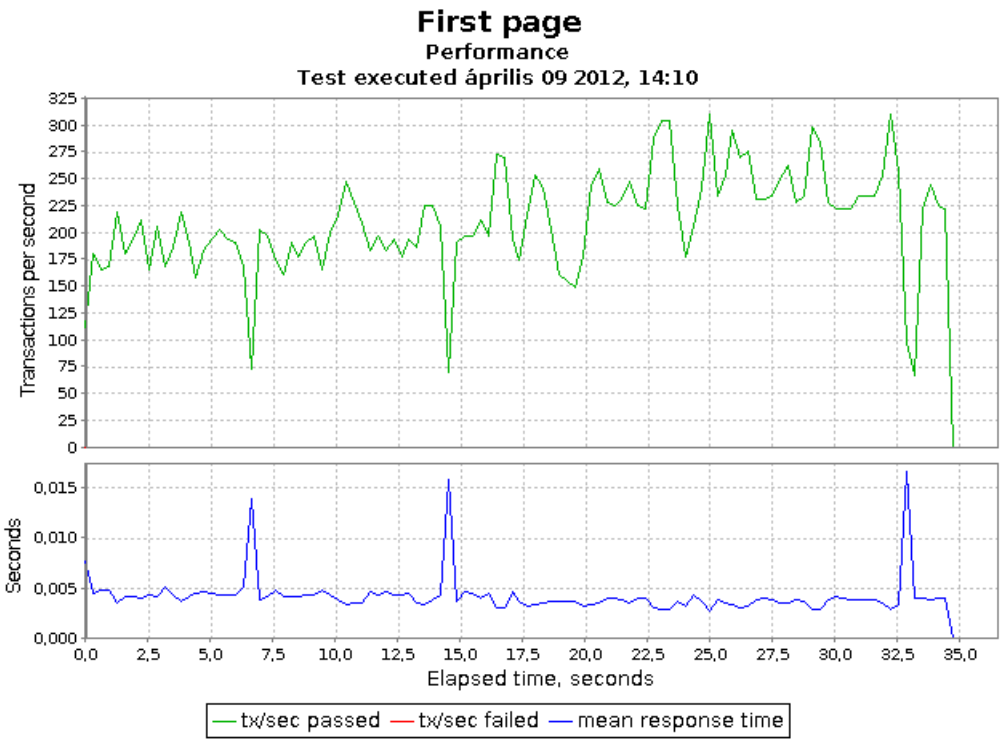
```
1  // 3-1. programlista: weblogic-http.py
2
3  # Recording many HTTP interactions as one test
4  #
5  # This example shows how many HTTP interactions can be grouped as a
6  # single test by wrapping them in a function.
7
8  from net.grinder.script.Grinder import grinder
9  from net.grinder.script import Test
10 from net.grinder.plugin.http import HTTPRequest
11 from HTTPClient import NVPair
12
13 # We declare a default URL for the HTTPRequest.
14 request = HTTPRequest(url = "http://localhost:7001")
15
```

```
16 def page1():
17     request.GET('/console')
18     request.GET('/console/login/LoginForm.jsp')
19     request.GET('/console/login/bea_logo.gif')
20
21 page1Test = Test(1, "First page").wrap(page1)
22
23 class TestRunner:
24     def __call__(self):
25         page1Test()
```

```
// 3-1. Naplórészlet

2012-04-09 14:10:35,087 INFO  csdev1-0 thread-0 [ run-7317, test-1 ]: http://localhost:7001/➥
    console/login/LoginForm.jsp -> 200 OK, 3161 bytes
2012-04-09 14:10:35,088 INFO  csdev1-0 thread-0 [ run-7317, test-1 ]: http://localhost:7001/➥
    console/login/bea_logo.gif -> 401 Unauthorized, 1518 bytes
2012-04-09 14:10:35,089 INFO  csdev1-0 thread-0 [ run-7318, test-1 ]: http://localhost:7001/➥
    console -> 302 Moved Temporarily, 255 bytes [Redirect, ensure the next URL is http://➥
    localhost:7001/console/]
2012-04-09 14:10:35,090 INFO  csdev1-0 thread-0 [ run-7318, test-1 ]: http://localhost:7001/➥
    console/login/LoginForm.jsp -> 200 OK, 3161 bytes
2012-04-09 14:10:35,091 INFO  csdev1-0 thread-0 [ run-7318, test-1 ]: http://localhost:7001/➥
    console/login/bea_logo.gif -> 401 Unauthorized, 1518 bytes
2012-04-09 14:10:35,092 INFO  csdev1-0 thread-0 [ run-7319, test-1 ]: http://localhost:7001/➥
    console -> 302 Moved Temporarily, 255 bytes [Redirect, ensure the next URL is http://➥
    localhost:7001/console/]
2012-04-09 14:10:35,098 INFO  csdev1-0 : received a stop message
2012-04-09 14:10:35,098 INFO  csdev1-0 thread-0 [ run-7319, test-1 ]: http://localhost:7001/➥
    console/login/LoginForm.jsp -> 200 OK, 3161 bytes
2012-04-09 14:10:35,099 INFO  csdev1-0 thread-0 [ run-7319, test-1 ]: http://localhost:7001/➥
    console/login/bea_logo.gif -> 401 Unauthorized, 1518 bytes
2012-04-09 14:10:35,101 INFO  csdev1-0 thread-0 [ run-7320 ]: shut down
...
finished 7320 runs
elapsed time is 34786 ms
```



3.1. ábra. Hálózati sávszélesség



3.2. ábra. TPS és átlagos válaszidő

Véletlenszám alapú tesztprogram

A 3-2. programlista bemutatja a tesztprogramok általános szerkezetét, ugyanis a *doIt()* helyére bármit írhatunk, mint tesztelő kód. Elöljáróban a 18. sorban lévő *tests* változó, mint lista létrehozásának működését értsük meg, ami szintén egy python lehetőség. A beépített *range()* függvény egy listát ad vissza, aminek az első elemét és a végét lehet megadni, ami azonban már nem eleme a gyűjteménynek. A példa jól szemlélteti, hogy ilyen módszerrel miképpen jön létre a 0–9 számok listája:

```
lista = [ i for i in range(0, 10) ]
print lista

Futás:
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

A 18. sorban ugyanez a nyelvi konstrukció használatos, de itt a kialakított lista nem egyszerűen a ciklusváltozó értékeiből, hanem *Test* objektumokból fog állni, azaz így 1 sorban 10 darabot is létre tudtunk hozni, amit a *tests* változón keresztül érünk el. Ugyanezen sorban azt is látjuk, hogy a tesztbe a szokásos módon – *wrap()* metódussal – be is csomagoltuk a tesztelés algoritmusát, ami a *doIt()*. Mit is csinál ő? Nézzük a 13-16 sorok közötti implementációját, de előtte vegyük észre, hogy egy véletlenszám generátort inicializálunk a 11. sorban, majd annak a következő értékeit mindig az *r* változón keresztül érhetjük majd el. Ezzel az eszközzel a [0.0, 1.0) balról zárt, jobbról nyílt tartományból lebegőpontos véletlenszámok kérhetők le. A használt *nextGaussian()* metódus ezt transzformálja a [-1.0, 1.0) intervallumra. Az alábbiakban látható erre egy egyszerű tesztprogram, ami modellezi a *doIt()* működését is! A visszaadott számok az 500-as érték körül változnak, amire az is hatással van, hogy a *list* változónak éppen melyik elemére hívtuk a *doIt()* metódust, azaz a

v paraméter értéke mekkora.

```
from java.util import Random
from java.lang import Math

r = Random()

def doIt(v):
    t = 500 + r.nextGaussian() * v * 10
    print int(t), t

list = [ i for i in range(0, 9) ]
for e in list:
    doIt( e )

Futás:
500 500.0
503 503.315333024
509 509.084804867
457 457.299418618
490 490.223846563
543 543.25559885
544 544.201668026
598 598.924652992
423 423.717224324
```

Visszatérve a *console.py* tesztprogram *doIt()* metódusához, annak működését már egyszerűen megérthetjük. Generál egy t véletlenszámot, majd egyszerűen elaltatja ennyivel a tesztfutást, ezzel szimulálva a teszt végrehajtási idejét. A *pass* a már ismert üres utasítás.

```
1  // 3-2. programlista: console.py
2
3  # Test script which generates some random data for testing the
4  # console.
5
6  from net.grinder.script.Grinder import grinder
7  from net.grinder.script import Test
8  from java.util import Random
9  from java.lang import Math
10
11 r = Random()
12
13 def doIt(v):
14     t = 500 + r.nextGaussian() * v * 10
15     grinder.sleep(int(t), 0)
16     pass
17
18 tests = [ Test(i, "Test %s" % i).wrap(doIt) for i in range(0, 10) ]
19
20 class TestRunner:
21     def __call__(self):
22 #         statistics = grinder.statistics
23
24 #         statistics.delayReports = 1
25
26         for test in tests:
27             test(test.__test__.number)
```

Végezetül nézzük meg a mindegyik Grinder teszt script részére szükséges tesztfuttató *TestRunner* osztály felépítését, amit a 20-27 sorok között tanulmányozhatunk. Csak emlékeztetőül jegyezzük meg, hogy a Grinder motor mindig ezt az osztályt példányosítja és úgy hívja meg, ahogy a pythonban egy objektumot függvényként hívunk. A korábbiakban elmagyaráztuk, hogy ehhez a *__call()__* metódus implementációja a követelmény, hiszen ennek a hívása jelenti az objektum futtatását, amiatt a 21-27 sorok között implementáltuk. A jobb megértés ér-

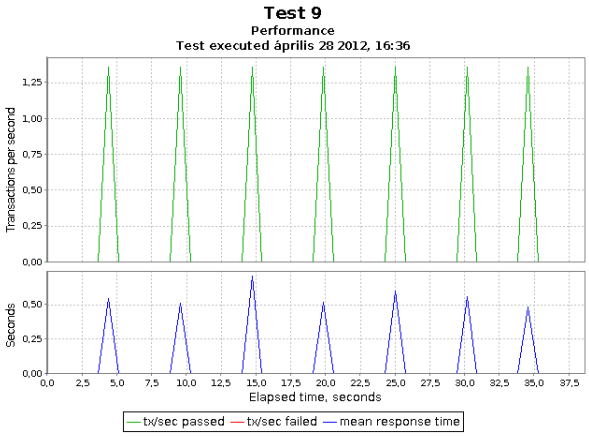
dekében a következő kis program ugyanúgy működik, ahogy a Grinder engine. Ugyanaz a kimenete, mint az előző kis tanuló programnak, de itt láthatjuk, hogy egy *MyTestRunner* osztályt, illetve annak *__call__()* metódusát miképpen írtuk meg, illetve a *myEngine* változót milyen módon hívtuk függvényként, átadva neki a 10 értéket. A 3-2. programlista 27. sorában a *test* egy *Test* osztálybeli objektum, hiszen a *tests* listának ilyen elemei vannak. Ráismerhetünk, hogy a *test* objektumot itt függvényként hívjuk. A paraméterezése a *doIt()* függvényével egyezik, mert azt csomagoltuk be korábban, ahol az átadott paraméter a teszt azonosító száma.

```
from java.util import Random
from java.lang import Math

r = Random()

def doIt(v, c):
    t = 500 + r.nextGaussian() * c * 10
    print int(t), t

list = [ i for i in range(0, 9) ]

class MyTestRunner:
    def __call__(self, c):
        for e in list:
            doIt( e, c )

myEngine=MyTestRunner()
myEngine( 10 )
```

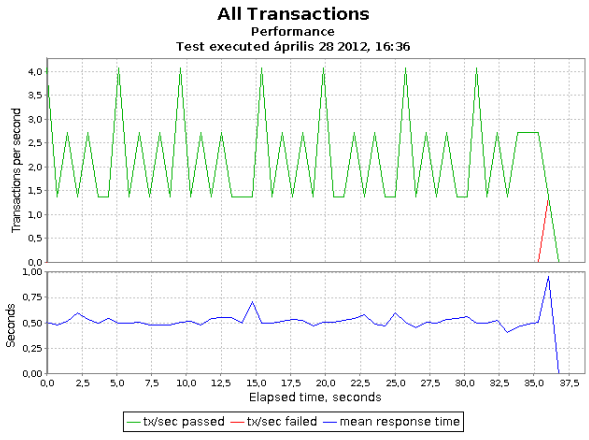
A 3-2. Naplórészlet a *console.py* tesztfutás naplójának részleteit mutatja. A napló elején lévő tesztfutások bejegyzéseit nagyrészt kihagytuk, mert azok a 3-25 sorok között megfigyelhető típusúak és azok ismétlődnek több száz soron keresztül. A 26. sorban láthatjuk, hogy a Grinder konzolról mikor küldtük ki a tesztelés befejezésének kérelmét (ez az érték = 16:37:06,565). A 3. sor alapján a tesztelést 16:36:29,343 időpillanatban kezdtük, de ha kíváncsiak vagyunk a tényleges nettó tesztidőre, akkor azt a 29. sorból olvashatjuk ki: elapsed time is 37244 ms. A 28. sor szerint ezen idő alatt 7 tesztet tudtunk elvégezni, melynek a teljes statisztikáját a 32-48 sorok mutatják. Mindezek grafikus elemzéséből két jellemző grafikont is betettünk, láthatjuk őket a 3.3. és 3.4. ábrákról. Az egyik csak a Test 9 esetet, a másik az összes teszt együttes vizsgálatát jeleníti meg.

```
1  // 3-2. Naplórészlet
2  ...
3  2012-04-28 16:36:29,343 INFO  csdev1-0 thread-0 [ run-0, test-0 ]: sleeping for 500 ms
4  2012-04-28 16:36:29,844 INFO  csdev1-0 thread-0 [ run-0, test-1 ]: sleeping for 497 ms
5  2012-04-28 16:36:30,343 INFO  csdev1-0 thread-0 [ run-0, test-2 ]: sleeping for 479 ms
6  2012-04-28 16:36:30,823 INFO  csdev1-0 thread-0 [ run-0, test-3 ]: sleeping for 513 ms
7  2012-04-28 16:36:31,337 INFO  csdev1-0 thread-0 [ run-0, test-4 ]: sleeping for 507 ms
8  2012-04-28 16:36:31,845 INFO  csdev1-0 thread-0 [ run-0, test-5 ]: sleeping for 597 ms
9  2012-04-28 16:36:32,443 INFO  csdev1-0 thread-0 [ run-0, test-6 ]: sleeping for 553 ms
10 2012-04-28 16:36:32,997 INFO  csdev1-0 thread-0 [ run-0, test-7 ]: sleeping for 513 ms
11 2012-04-28 16:36:33,511 INFO  csdev1-0 thread-0 [ run-0, test-8 ]: sleeping for 498 ms
12 2012-04-28 16:36:34,010 INFO  csdev1-0 thread-0 [ run-0, test-9 ]: sleeping for 544 ms
13 2012-04-28 16:36:34,556 INFO  csdev1-0 thread-0 [ run-1, test-0 ]: sleeping for 500 ms
14 2012-04-28 16:36:35,057 INFO  csdev1-0 thread-0 [ run-1, test-1 ]: sleeping for 493 ms
15 2012-04-28 16:36:35,551 INFO  csdev1-0 thread-0 [ run-1, test-2 ]: sleeping for 498 ms
16 2012-04-28 16:36:36,050 INFO  csdev1-0 thread-0 [ run-1, test-3 ]: sleeping for 526 ms
17 2012-04-28 16:36:36,577 INFO  csdev1-0 thread-0 [ run-1, test-4 ]: sleeping for 484 ms
18 2012-04-28 16:36:37,065 INFO  csdev1-0 thread-0 [ run-1, test-5 ]: sleeping for 479 ms
19 2012-04-28 16:36:37,545 INFO  csdev1-0 thread-0 [ run-1, test-6 ]: sleeping for 480 ms
20 2012-04-28 16:36:38,026 INFO  csdev1-0 thread-0 [ run-1, test-7 ]: sleeping for 483 ms
21 2012-04-28 16:36:38,510 INFO  csdev1-0 thread-0 [ run-1, test-8 ]: sleeping for 474 ms
22 2012-04-28 16:36:38,985 INFO  csdev1-0 thread-0 [ run-1, test-9 ]: sleeping for 507 ms
23 ...
24 2012-04-28 16:37:05,609 INFO  csdev1-0 thread-0 [ run-7, test-1 ]: sleeping for 499 ms
25 2012-04-28 16:37:06,109 INFO  csdev1-0 thread-0 [ run-7, test-2 ]: sleeping for 528 ms
26 2012-04-28 16:37:06,565 INFO  csdev1-0 : received a stop message
27 2012-04-28 16:37:06,568 INFO  csdev1-0 thread-0 [ run-7 ]: shut down
28 2012-04-28 16:37:06,568 INFO  csdev1-0 thread-0: finished 7 runs
29 2012-04-28 16:37:06,572 INFO  csdev1-0 : elapsed time is 37244 ms
```

```
2012-04-28 16:37:06,572 INFO  csdev1-0 : Final statistics for this process:
2012-04-28 16:37:06,586 INFO  csdev1-0 :
             Tests       Errors      Mean Test Time (ms)   Test Time Standard Deviation (ms)   TPS
Test 0       8           0           500,62                0,70                               0,21   "Test 0"
Test 1       8           0           499,62                6,84                               0,21   "Test 1"
Test 2       7           1           487,86                20,95                              0,19   "Test 2"
Test 3       7           0           520,14                34,90                              0,19   "Test 3"
Test 4       7           0           509,71                23,19                              0,19   "Test 4"
Test 5       7           0           517,43                59,07                              0,19   "Test 5"
Test 6       7           0           526,57                39,21                              0,19   "Test 6"
Test 7       7           0           495,00                34,26                              0,19   "Test 7"
Test 8       7           0           492,00                24,51                              0,19   "Test 8"
Test 9       7           0           558,29                70,03                              0,19   "Test 9"

Totals       72          1           510,43                41,81                              1,93
```



3.3. ábra. A Test 9 teljesítmény grafikonja



3.4. ábra. Az összes teszt együttes teljesítmény grafikonja

HTTP Cookie tesztprogram

A további példáinkban általában már nem fogjuk bemutatni a tesztfutásokat, jellemzően csak igyekszünk elmagyarázni, hogy a bemutatott teszt script mit csinál, így az olvasó további ötleteket meríthet saját scriptjeinek elkészítéséhez. Ebből a példából a HTTP cookie tesztelési lehetőségét ismerhetjük meg. A *HTTPClient* library-t fogjuk használni, ami támogatja a sütikkel való manipulációkat, amihez a *CookiePolicyHandler* class alkalmazása is fontos. Ez utóbbi osztályt definiálja majd a 3-3. programlista *cookies.py* scriptje is. A példa a cookie tárolását, hozzáadását és törlését is bemutatja. A 18-25 sorok közötti *MyCookiePolicyHandler* class őse a *CookiePolicyHandler* osztály, így könnyen építhetünk saját kezelőt. Most csak 2 metódust írunk felül, azaz megmondjuk a *acceptCookie()* és *sendCookie()* speciális működését. A kódok meglehetősen egyszerűek, kizárólag csak naplózzák ezeket az eseményeket (20. és 24. sorok). A 27. sor beállítja az eseménykezelést a most megalkotott osztályunk egy objektumára, ugyanis a paraméterként átadott *MyCookiePolicyHandler()* egy konstruktor hívás, így eredményeképpen egy névtelen objektum jön létre. A *test1* nevű *Test* objektumot a 29-30 sorok között definiáltuk, most csak 1 van belőle és a tesztelő algoritmusa a *HTTPRequest()* objektum függvényként való hívása lesz. Ezzel minden előkészítettünk, hogy a *TestRunner* $\_\_call\_\_()$ metódusát elkészíthessük (34-64 sorok).

```
1  // 3-3. programlista: cookies.py
2
3  # HTTP cookies
4  #
5  # If you really want direct control over the cookie headers, you
6  # can disable the automatic cookie handling with:
7  #    HTTPPluginControl.getConnectionDefaults().useCookies = 0
8
9  from net.grinder.script.Grinder import grinder
10 from net.grinder.script import Test
11 from net.grinder.plugin.http import HTTPRequest, HTTPPluginControl
12 from HTTPClient import Cookie, CookieModule, CookiePolicyHandler
13 from java.util import Date
14
15 log = grinder.logger.info
16
17 # Set up a cookie handler to log all cookies that are sent and received.
18 class MyCookiePolicyHandler(CookiePolicyHandler):
19     def acceptCookie(self, cookie, request, response):
20         log("accept cookie: %s" % cookie)
21         return 1
22
23     def sendCookie(self, cookie, request):
24         log("send cookie: %s" % cookie)
25         return 1
26
27 CookieModule.setCookiePolicyHandler(MyCookiePolicyHandler())
28
29 test1 = Test(1, "Request resource")
30 request1 = test1.wrap(HTTPRequest())
31
```

```
32
33 class TestRunner:
34     def __call__(self):
35         # The cache of cookies for each  worker thread will be reset at
36         # the start of each run.
37
38         result = request1.GET("http://localhost:7001/console/?request1")
39
40         # If the first response set any cookies for the domain,
41         # they willl be sent back with this request.
42         result2 = request1.GET("http://localhost:7001/console/?request2")
43
44         # Now let's add a new cookie.
45         threadContext = HTTPPluginControl.getThreadHTTPClientContext()
46
47         expiryDate = Date()
48         expiryDate.year += 10
49
50         cookie = Cookie("key", "value","localhost", "/", expiryDate, 0)
51
52         CookieModule.addCookie(cookie, threadContext)
53
54         result = request1.GET("http://localhost:7001/console/?request3")
55
56         # Get all cookies for the current thread and write them to the log
57         cookies = CookieModule.listAllCookies(threadContext)
58         for c in cookies: log("retrieved cookie: %s" % c)
59
60         # Remove any cookie that isn't ours.
61         for c in cookies:
62             if c != cookie: CookieModule.removeCookie(c, threadContext)
63
64         result = request1.GET("http://localhost:7001/console/?request4")
```

A 38. és 42. sorok HTTP kéréseinek hatását olvassuk el a script-ben hozzájuk fűzött megjegyzésekből! A 45-52 sorok egy új cookie létrehozását mutatják, ami amiatt az 54. sor HTTP kérésénél már el fog menni a kérésben. Az 57-58 sorokban naplózzuk az összes ide tartozó sütit, majd a 61-62 sorokban mindegyiket kitöröljük. Ezzel azt érjük el, hogy a 64. sorban lévő request már egyetlen sütit sem fog küldeni a kéréssel.

Form alapú hitelesítés tesztprogram

A HTTP alapú hitelesítéseket (basic, form, certificate) ismertnek tételezzük fel, azokat itt most nem tudjuk áttekinteni. A 3-4. programlista a form alapú hitelesítési lépésekre tartalmaz egy tesztprogramot, így aki ezt nem ismeri mindenképpen nézze át a példa elolvasása előtt. Amikor egy nem hitelesített user próbál elérni egy védett URL-en lévő erőforrást, akkor a jogosultságkezelés alapja a felhasználó azonosságának az ismerete. Amennyiben a hitelesítés form alapú, úgy a webszerver egy login lapra irányítja át a kérést, de megjegyzi az eredeti erőforrásra hivatkozó URL-t is. Ez a lap fogja elküldeni (HTTP POST) a *username* és *password* értékeket egy speciális *j_security_check* action részére. A 8. és 9. sorok egy-egy teszt objektumot határoznak

meg az erőforrás lekérésére, illetve a hitelesítési információk (credential) elküldésére. A 11-20 sorok között kifejtett *TestRunner* 13. sorában a *request* hivatkozik a becsomagolt *HTTPRequest(...)* függvényhívásra, ami az első teszt. A másik teszt *wrap*-pelése a 31. sorban történik majd, a *maybeAuthenticate()* metódus belsejében. Emiatt nézzük meg előbb ezt a függvénytagot, amely a 22-34 sorok közé került. A paraméterként kapott *lastResult* természetesen a megelőző request eredménye lesz, így amennyiben a *statusCode* értéke ezt indokolja végrehajtja a hitelesítést (username=weblogic, password=weblogic), majd a 34. sorban az eredeti URL-re posztólja vissza a kérést. Ennyi előkészítés után a *TestRunner* megértése már nagyon egyszerű lesz. A 13. sor becsomagolásáról már említést tettünk, a 16. sor viszont el is végzi a konkrét HTTP kérést. A 18. sorban elegánsan használjuk a megírt segédmetódusunkat, majd ezután a 20. sor hívása már sikeres lesz, ugyanis akkor már a HTTP session tartalmazni fogja a szükséges security tokent. A form alapú hitelesítésről javasoljuk elolvasni a következő összefoglalót: `http://sling.apache.org/site/form-based-authenticationhandler.html`.

```
1  // 3-4. programlista: fba.py (Form Bases hitelesítés)
2
3  from net.grinder.script.Grinder import grinder
4  from net.grinder.script import Test
5  from net.grinder.plugin.http import HTTPRequest
6  from HTTPClient import NVPair
7
8  protectedResourceTest = Test(1, "Request resource")
9  authenticationTest = Test(2, "POST to j_security_check")
10
11 class TestRunner:
12     def __call__(self):
13         request = protectedResourceTest.wrap(
14             HTTPRequest(url="http://localhost:7001/console"))
15
16         result = request.GET()
17
18         result = maybeAuthenticate(result)
19
20         result = request.GET()
21
22 def maybeAuthenticate(lastResult):
23     if lastResult.statusCode == 401 \
24     or lastResult.text.find("j_security_check") != -1:
25
26         grinder.logger.info("Challenged, authenticating")
27
28         authenticationFormData = ( NVPair("j_username", "weblogic"),
29                                    NVPair("j_password", "weblogic"),)
30
31         request = authenticationTest.wrap(
32             HTTPRequest(url="%s/j_security_check" % lastResult.originalURI))
33
34         return request.POST(authenticationFormData)
```

Adatbázis alapú tesztprogram

Az adatbázisok megfelelő teljesítményének kérdését nem lehet elég alkalommal kiemelni, most megtanuljuk ezt mérni is. Példánkban (3-5. programlista) egy Oracle adatbázist teszteltünk, amihez a thin driver-t használtuk (ennek a CLASSPATH-on kell lennie). A 8. és 9. sorok 2 *Test* típusú változója (*test1* és *test2*) már sejteti, hogy az SQL INSERT és SELECT műveletekre végzünk teszteket. A tesztelésnél kihasználjuk, hogy jython a futtató környezet, így a Java JDBC apparátust tudjuk használni. Ennek megfelelően a 12. sorban regisztráljuk is a Java *OracleDriver*-t. A 14-16 sorok egy metódusba teszik az adatbázishoz kapcsolódás műveletét, míg a 18-20 sorok pedig a lekapcsolódást. Ez utóbbiban az *object* paraméter egy Java *Connection* vagy *Statement* osztálybeli objektum is lehet, mert mindkettőnek van *close()* metódusa. A 23. sor az adatbázishoz való kapcsolódás, míg a 24. egy SQL *statement* objektum létrehozását jelenti. Ez utóbbi teszi lehetővé az SQL parancsok kiadását, ahogy azt a 26. sorban látjuk is. Ezután a 29. sorban létrehozunk egy *grinder_fun* nevű táblát, látható, hogy 2 oszlopa van (*thread*, *run*). A 31-32 sorokban lezárjuk a statement-et és a connection-t is. Mindezen lépések a tesztelés előkészítésére szolgálnak, azaz legyen egy üres *grinder_fun* nevű táblánk. Most pedig nézzük meg, hogy maga a tesztelés mit jelent, azaz a *TestRunner* mit csinál! A kód nagyon egyszerű, a *test1* insert-eket ad ki, amihez inputként a *threadNumber* és *runNumber* értékeket használja. Nincs ebben semmi ismeretlen. A *test2* ugyanezen a táblán select-et ad ki, ez is könnyen érthető. Egy tesztfutás végén pedig bezárjuk az erőforrásokat (52-53 sorok).

```
1  // 3-5. programlista: jdbc.py
2
3  from java.sql import DriverManager
4  from net.grinder.script.Grinder import grinder
5  from net.grinder.script import Test
6  from oracle.jdbc import OracleDriver
7
8  test1 = Test(1, "Database insert")
9  test2 = Test(2, "Database query")
10
11 # Load the Oracle JDBC driver.
12 DriverManager.registerDriver(OracleDriver())
13
14 def getConnection():
15     return DriverManager.getConnection(
16         "jdbc:oracle:thin:@127.0.0.1:1521:mysid", "wls", "wls")
17
18 def ensureClosed(object):
19     try: object.close()
20     except: pass
21
22 # One time initialisation that cleans out old data.
23 connection = getConnection()
24 statement = connection.createStatement()
25
26 try: statement.execute("drop table grinder_fun")
27 except: pass
28
```

```
29 statement.execute("create table grinder_fun(thread number, run number)")
30
31 ensureClosed(statement)
32 ensureClosed(connection)
33
34 class TestRunner:
35     def __call__(self):
36         connection = None
37         statement = None
38
39         try:
40             connection = getConnection()
41             statement = connection.createStatement()
42
43             testInsert = test1.wrap(statement)
44             testInsert.execute("insert into grinder_fun values(%d, %d)" %
45                                (grinder.threadNumber, grinder.runNumber))
46
47             testQuery = test2.wrap(statement)
48             testQuery.execute("select * from grinder_fun where thread=%d" %
49                               grinder.threadNumber)
50
51         finally:
52             ensureClosed(statement)
53             ensureClosed(connection)
```

JMS Sender alapú tesztprogram

A most bemutatandó tesztprogram (3-6. programlista) a Java Message Queue-ra (röviden: JMSQ) történő írást modellezi. Mindegyik worker szál létrehoz egy queue session-t, küld a queue-ra 10 darab üzenetet, majd bezárja a kapcsolatot. Mindehhez az Oracle Weblogic környezetet fogjuk használni, így eközben látni fogjuk azokat a technikai lépéseket is, amiket ebben a konténerben kell mindehhez elvégeznünk. Ilyen lesz például a JNDI[1] fához való hozzáférés. A tesztelésnél a CLASSPATH-on kell lennie a *weblogic.jar* file-nak. A 16. sor a hálózaton keresztül is működő JNDI kontextus objektumot (neve: *initialContext*) szerzi meg, amihez a 12-14 sorokban felépített *properties* objektumot használja. A távoli szerveren lévő *weblogic.examples.jms.exampleQueue* nevű queue-ra szeretnénk majd írni, ezért a 18. sorban megszerezzük a *connectionFactory*-t, majd ezután a távoli Queue objektumra egy referenciát kérünk le, amit a 19. sorban a *queue* változóban fogunk tárolni. Ezután már nincs szükségünk a JNDI elérésre, ezért azt lezárjuk. A 23. sorban a *connectionFactory* segítségével létrehozunk egy kapcsolatot a távoli JMS szerverhez és el is indítjuk ennek a használatát a *start()* metódussal. A későbbiekben véletlen számokat dobunk majd a JMSQ-ra, ezért a már ismert módszerrel létrehozunk egy *Random* típusú *random* változót. A 28-33 sorok közötti *createBytesMessage()* metódus generálja a JMSQ-ra dobott tényleges tartalmakat, felhasználva a véletlenszám generálást, majd a 31-32 sorokban a JMS API üzenetlétrehozási metódusát, amely üzenetet a végén visszaadja ez a metódus. Végeztünk a szokásos előkészítésekkel, jöhet ismét a *TestRunner* megírása, őt láthatjuk a 35-54 sorok között. Ennyi

[1] JNDI=Java Naming and Directory Interface

munka után ezt elkészíteni már nem túl nehéz, nézzük csak! Aki ismeri a Java JMS API-t, annak minden sor ismerős, aki pedig nem az próbálja először azt áttekinteni. A 40. sorban a szokásos módon hozzuk létre a *session* változót, hogy abból és a korábban már elkészített *queue* változóból egy *sender*-t készíthessünk. Az *instrumentedSender* lesz az egyetlen *Test* változónk, ebbe csomagoljuk be a *sender* objektumot is. A 45. sorban legyártjuk az üzenetet (*message*), majd egy for ciklussal azt 10 alkalommal rádobjuk a queue-ra. Közben mindig tartunk egy rövid szünetet is. A teszt a *session* bezárásával fejeződik be.

```
1  // 3-6. programlista: jmssender.py
2
3  from net.grinder.script.Grinder import grinder
4  from net.grinder.script import Test
5  from jarray import zeros
6  from java.util import Properties, Random
7  from javax.jms import Session
8  from javax.naming import Context, InitialContext
9  from weblogic.jndi import WLInitialContextFactory
10
11 # Look up connection factory and queue in JNDI.
12 properties = Properties()
13 properties[Context.PROVIDER_URL] = "t3://localhost:7001"
14 properties[Context.INITIAL_CONTEXT_FACTORY] = WLInitialContextFactory.name
15
16 initialContext = InitialContext(properties)
17
18 connectionFactory = initialContext.lookup("weblogic.examples.jms.➥
       QueueConnectionFactory")
19 queue = initialContext.lookup("weblogic.examples.jms.exampleQueue")
20 initialContext.close()
21
22 # Create a connection.
23 connection = connectionFactory.createQueueConnection()
24 connection.start()
25
26 random = Random()
27
28 def createBytesMessage(session, size):
29     bytes = zeros(size, 'b')
30     random.nextBytes(bytes)
31     message = session.createBytesMessage()
32     message.writeBytes(bytes)
33     return message
34
35 class TestRunner:
36     def __call__(self):
37         log = grinder.logger.info
38
39         log("Creating␣queue␣session")
40         session = connection.createQueueSession(0, Session.AUTO_ACKNOWLEDGE)
41
```

```
42          sender = session.createSender(queue)
43          instrumentedSender = Test(1, "Send a message").wrap(sender)
44
45          message = createBytesMessage(session, 100)
46
47          log("Sending ten messages")
48
49          for i in range(0, 10):
50              instrumentedSender.send(message)
51              grinder.sleep(100)
52
53          log("Closing queue session")
54          session.close()
```

A Grinder statisztika API

A Grinder ezen API-ja lehetővé teszi, hogy új statisztikákkal egészítsük ki méréseinket. Mindegyik statisztika egy egyedi névvel rendelkezik. Az alap statisztikák (Basic statistics) egy *long* vagy *double* értéket jelentenek. A minták statisztikája (Sample statistics) pedig egy összegzett (aggregated) *long* vagy *double* értékeket jelentenek. Ennek van 3 gyakran használt esete a *count (*a minták száma*)*, a *sum (*a minta értékeinek összege*)* és a *variance* (a minta varianciája). Nézzük milyen statisztikák vannak:

- *errors*: A hibával végződött tesztelések száma.
- *timedTests*: A jól elvégzett tesztek, azaz ez egy mintastatisztika.
- *userLong0, userLong1, userLong2, userLong3, userLong4*: Saját célú statisztika, ahol *long* a mért típus.
- *userDouble0, userDouble1, userDouble2, userDouble3, userDouble4*: Saját célú statisztika, ahol *double* a mért típus.
- *untimedTests*: Sikeres tesztek, de nem mértünk időzítés információt.

A következő példában bemutatjuk hogyan lehet használni a statisztikai apparátust. Ott a következő eszközöket fogjuk majd használni a *grinder.statistics* csomagból:

- *registerDataLogExpression(displayName, expression)* metódus: Egy új részletező statisztikát regisztrál.
- *registerSummaryExpression(displayName, expression)* metódus: Egy új összegző statisztikát regisztrál.
- *forCurrentTest*: Beállítható ezzel, hogy a mérést mire gyűjtjük.

Az expression string a statisztikai nevekből és műveleti jelekből áll, amiket postfix módon kell leírnunk. Példa: *(/ (sum timedTests) (count timedTests))* jelentése a teszt 1 futásának átlagos ideje milliszekundumban. A statisztika API további részleteiről itt olvashatunk: `http://grinder.sourceforge.net/g3/statistics.html`.

JMS Receiver alapú tesztprogram

Most végezzük el az előző teszt fordítottját, azaz olvassunk a queue-ról! Ennek a módszerét részben demonstrálta a 3-6. programlista, aminek bemutatásánál már nem írjuk le újra azokat a lépéseket (egészen a 25. sorig), amiket korábban megismertünk. A program létrehozza a JMS

session-t, olvas 10 üzenetet a már ismert queue-ról, majd lezárja a kapcsolatot. További érdekesség lesz, hogy a *jmsreceiver.py* új elemként bemutatja a *Grinder statistics API* használatát is, ahol a *delivery time* egy testre szabott értékfigyelés lesz. Ennek megfelelően a 33. sorban rögzítünk egy *userLong0* nevű statisztikát, ami a riportokban *Delivery time* néven fog megjelenni. A 34. sorban pedig egy összegző statisztikát regisztrálunk, ahol a kiszámítás módját postfix alakban adtuk meg. A jelentése az, hogy adja össze az időzített és nem időzített sikeres tesztek számát (ezt a Grinder méri, előre definiálva van), majd ezzel ossza el a korábban mért *userLong0* értéket. A 40-41 sorok között definiált *recordDeliveryTime()* függvény rendeli a *deliveryTime* idő értékét (ezt a 67. sorban megadott módon tudjuk megszerezni) a *userLong0*-hoz. A scriptben a 43. sorban adtuk meg a mérés egyetlen *Test* osztálybeli objektumát *recordTest* néven, ami már az említett *recordDeliveryTime()* függvényt csomagolja be, azaz ez lesz a teszt algoritmusa. Ahogy azt már megszokhattuk, befejezésül nézzük meg a *TestRunner* class felépítését. A 47-49 sorok közötti *__init__()* az osztály konstruktora, most az inicializálási feladatok miatt ezt is elkészítettük. Tekintettel arra, hogy most üzeneteket fogadunk, így az 57-58 sorokban most a *receiver* objektumot kellett létrehozni a JMS API-nál ismert módon. Az 58. sorban beállítjuk magunkra (*self*) a *messageListener*-t, azaz működni fog a 85-96 sorok közötti *onMessage()* metódus, ez fogja egymás után levenni az üzeneteket, kiszámítja a *deliveryTime* értékét (91. sor) és eltárolja. Mindeközben van egy *Condition* objektum, ami a szálak kezelését támogatja, segítségével most szinkronizálunk a mérés algoritmusa és az asszinkron *onMessage()* között. A már említett becsomagolt *recordTest* nevű *Test* objektumot a 74. sorban futtatjuk.

```
1  // 3-7. programlista: jmsreceiver.py
2
3  from java.lang import System
4  from java.util import Properties
5  from javax.jms import MessageListener, Session
6  from javax.naming import Context, InitialContext
7  from net.grinder.script.Grinder import grinder
8  from net.grinder.script import Test
9  from threading import Condition
10 from weblogic.jndi import WLInitialContextFactory
11
12 # Look up connection factory and queue in JNDI.
13 properties = Properties()
14 properties[Context.PROVIDER_URL] = "t3://localhost:7001"
15 properties[Context.INITIAL_CONTEXT_FACTORY] = WLInitialContextFactory.name
16
17 initialContext = InitialContext(properties)
18
19 connectionFactory = initialContext.lookup("weblogic.examples.jms.➥
       QueueConnectionFactory")
20 queue = initialContext.lookup("weblogic.examples.jms.exampleQueue")
21 initialContext.close()
22
23 # Create a connection.
24 connection = connectionFactory.createQueueConnection()
25 connection.start()
```

```
26
27 # Add two statistics expressions:
28 # 1. Delivery time:- the mean time taken between the server sending
29 #    the message and the receiver receiving the message.
30 # 2. Mean delivery time:- the delivery time averaged over all tests.
31 # We use the userLong0 statistic to represent the "delivery time".
32
33 grinder.statistics.registerDataLogExpression("Delivery time", "userLong0")
34 grinder.statistics.registerSummaryExpression(
35               "Mean delivery time",
36                         "(/ userLong0(+ timedTests untimedTests))")
37
38 # We record each message receipt against a single test. The
39 # test time is meaningless.
40 def recordDeliveryTime(deliveryTime):
41     grinder.statistics.forCurrentTest.setValue("userLong0", deliveryTime)
42
43 recordTest = Test(1, "Receive messages").wrap(recordDeliveryTime)
44
45 class TestRunner(MessageListener):
46
47     def __init__(self):
48         self.messageQueue = []          # Queue of received messages not yet ➥
               recorded.
49         self.cv = Condition()           # Used to synchronise thread activity.
50
51     def __call__(self):
52         log = grinder.logger.info
53
54         log("Creating queue session and a receiver")
55         session = connection.createQueueSession(0, Session.AUTO_ACKNOWLEDGE)
56
57         receiver = session.createReceiver(queue)
58         receiver.messageListener = self
59
60         # Read 10 messages from the queue.
61         for i in range(0, 10):
62
63             # Wait until we have received a message.
64             self.cv.acquire()
65             while not self.messageQueue: self.cv.wait()
66             # Pop delivery time from first message in message queue
67             deliveryTime = self.messageQueue.pop(0)
68             self.cv.release()
69
70             log("Received message")
71
72             # We record the test a here rather than in onMessage
73             # because we must do so from a worker thread.
74             recordTest(deliveryTime)
75
76         log("Closing queue session")
```

```
77          session.close()
78
79          # Rather than over complicate things with explict message
80          # acknowledgement, we simply discard any additional messages
81          # we may have read.
82          log("Received %d additional messages" % len(self.messageQueue))
83
84      # Called asynchronously by JMS when a message arrives.
85      def onMessage(self, message):
86          self.cv.acquire()
87
88          # In WebLogic Server JMS, the JMS timestamp is set by the
89          # sender session. All we need to do is ensure our clocks are
90          # synchronised...
91          deliveryTime = System.currentTimeMillis() - message.getJMSTimestamp()
92
93          self.messageQueue.append(deliveryTime)
94
95          self.cv.notifyAll()
96          self.cv.release()
```

E-Mail alapú tesztprogram

A levélküldő rendszerünk teljesítményének egy lehetséges mérését valósítja a 3-8. programlista. A háttérben a JavaMail API-t használjuk, így az ahhoz szükséges jar file-oknak elérhetőnek kell lenniük. Akit ez a téma jobban érdekel, olvassa el ezt a dokumentumot: `http://www.oracle.com/technetwork/java/javamail-1-149769.pdf`. A tesztprogramunk egyébként kiváló eszköz lehet SPAM-ek nagyszámú küldésére, így erre ezt ne használjuk! A script szerzőit a példaprogram elején lévő megjegyzésből tudhatjuk meg. A script elég egyszerű felépítésű, az egyetlen Grinder *Test* objektuma (neve: *emailSendTest1*) a 15. sorban lett létrehozva és már nézhetjük is a *TestRunner* által elvégzett feladatot. A 19. sornál a saját mail szerverünkre kell állítani az *smtpHost* változót. Ez a gmail esetén például *smtp.gmail.com* érték.

A 26-34 sorok között az elküldendő MIME message formátumnak megfelelő levélüzenetet állítjuk össze, ahogy láthatjuk ez egy generált tartalom. A 37-43 sorok között pedig a levél fizikai elküldése valósul meg. A Java levélkezelő API a system property beállításokból veszi ki a levelező szerver nevét, emiatt szükséges a *mail.smtp.host* Java környezeti változót a 22. sorban a célszerverre állítani. A 23. sorban szerezzük meg a *session* objektumot, amit a levél összeállítása és elküldése (transport) során is használunk. A message tárgy (subject) részébe itt is – hasonlóan az adatbázis-kezelés részhez – a *runNumber* és *threadNumber* értékeket tesszük be. A *setFrom()* helyes kitöltése mindig fontos, ha hitelesítés kell a levélküldéshez. Itt a 40. sor *connect()* metódushívása hitelesítést is kér, emiatt ez most is fontos. A levél szövege egy konstans tartalom lesz.

```
1  // 3-8. programlista: email.py
2
3  #
4  # Copyright (C) 2004 Tom Pittard
5  # Copyright (C) 2004-2008 Philip Aston
```

```
6  # Distributed under the terms of The Grinder license.
7
8  from net.grinder.script.Grinder import grinder
9  from net.grinder.script import Test
10
11 from java.lang import System
12 from javax.mail import Message, Session
13 from javax.mail.internet import InternetAddress, MimeMessage
14
15 emailSendTest1 = Test(1, "Email Send Engine")
16
17 class TestRunner:
18     def __call__(self):
19         smtpHost = "mailhost"
20
21         properties = System.getProperties()
22         properties["mail.smtp.host"] = smtpHost
23         session = Session.getInstance(System.getProperties())
24         session.debug = 1
25
26         message = MimeMessage(session)
27         message.setFrom(InternetAddress("TheGrinder@yourtestdomain.net"))
28         message.addRecipient(Message.RecipientType.TO,
29                              InternetAddress("you@yourtestdomain.net"))
30         message.subject = "Test email %s from thread %s" % (grinder.runNumber,
31                                                             grinder.threadNumber)
32
33         # One could vary this by pointing to various files for content
34         message.setText("SMTPTransport Email works from The Grinder!")
35
36         # Wrap transport object in a Grinder Jython Test Wrapper
37         transport = emailSendTest1.wrap(session.getTransport("smtp"))
38
39         transport = emailSendTest1.wrap(transport)
40         transport.connect(smtpHost, "username", "password")
41         transport.sendMessage(message,
42                               message.getRecipients(Message.RecipientType.TO))
43         transport.close()
```

Amennyiben a gmail rendszert akarjuk használni, úgy a proprty értékek ezek legyenek:

```
Properties props = new Properties();
props.put("mail.smtp.auth", "true");
props.put("mail.smtp.starttls.enable", "true");
props.put("mail.smtp.host", "smtp.gmail.com");
props.put("mail.smtp.port", "587");
```

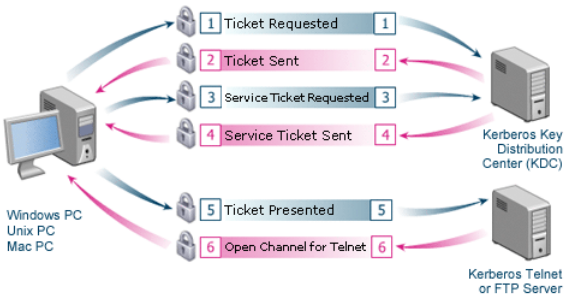
Látható, hogy itt a megszokott titkosított SSL csatornát használjuk. A session lekérése pedig így történhet:

```
Session session = Session.getInstance(props,
 new javax.mail.Authenticator() {
        protected PasswordAuthentication ➥
            getPasswordAuthentication() {
        return new PasswordAuthentication(➥
            username, password);
        }
 });
```

4. A Java biztonsági rendszere - Kerberos alapú SSO

A Kerberos egy régóta létező *Computer Network Authentication Protocol* amit a MIT-en (*Massachusetts Institute of Technology*) fejlesztettek ki. A név a görög mitológiában szereplő 3 fejű kutyára utal, amit még Cerberus néven is emlegetnek. A 4.1 ábrán látható KDC, Szerver és Kliens, mint az együttműködés 3 oldala ihlette az elnevezést. A cikk egy Windows infrastruktúrát használó Java szerver környezetben mutatja be a Kerberos használatát, szakítva a Windows NTLM v1/v2-re épülő megközelítéssel.

A Kerberos áttekintése



4.1. ábra. A Kerberos működése

A mai Windows környezetek használata közben többször botlunk a Kerberos névbe. Mi ez valójában? A Windows 2000 óta Ő a Windows alapértelmezett hitelesítő protokollja. A Kerberos biztonsági protokollt az 1980-as években fejlesztették ki az USA-ban. Jelenleg elterjedt 5. verziója az IETF (*Internet Engineering Task Force*) RFC 1510 szabványban van specifikálva. A kompatibilitás megőrzése érdekében a Windows infrastruktúrák még elterjedten használják a régebbi hitelesítési és biztonsági eljárásokat is. Ilyen az NTLM v1/v2, de a fő protokollá a Kerberos vált. Az RFC 1510 szabvány pontosan meghatározza azt is, hogy milyen egyéb biztonsági szolgáltatásokkal tud a Kerberos együttműködni. Feladata a felhasználók hitelesítése a hálózati szolgáltatások eléréséhez. Támogatja, hogy a hitelesítési kommunikáció során az adatok titkosítva haladjanak át a hálózaton, továbbá harmadik személy ne tudja elolvasni vagy módosítani azokat. Az operációs rendszerben rendelkezésre álló kódolási eljárásokat lehet használni, de a titkosítás – ha lehetséges – az úgynevezett titkos kulcsú (*secret key encryption*) kódolási algoritmus szerint történik. Ilyenkor egy titkos kulccsal megy végbe az adatok kódolása a küldő oldalon, a fogadónál pedig ugyanarra a kulcsra van szükség a visszafejtéshez. A küldő biztos lehet benne, hogy az információt csak a fogadó tudja értelmezni, a fogadó pedig megbízhat abban, hogy az információ a küldőtől érkezett. A felhasználói fiókok és jelszavak a KDC-ben (*Key Distribution Center*=Kulcskiosztó Központ) tárolódnak. Ez a Windows szerver környezetben a tartományvezérlőkön, az *Active Directory*-ban kapott helyet és ott erős titkosítással vannak védve az adatok. A protokoll a többféle titkosítási algoritmuson kívül a különböző hosszúságú titkosítási kulcsokat is megérti. Használhatjuk például a DES (*Data Encryption Standard*) RC4 algoritmusának a 128 bites verzióját is.

A Kerberos jegyrendszer

A jegyrendszer működése a háttérben, láthatatlanul zajlik. A felhasználó annyit vesz észre belőle, hogy kéri az operációs rendszer a felhasználói nevét és jelszavát, ő ezt megadja és elérhetővé válnak számára a jogosultságának megfelelő erőforrások. A KDC rendelkezésre állása magas. A 4.1. ábra mutatja a hitelesítés kommunikációs

szekvenciáját, illetve a közben használt security jegyeket (*ticket*). A hitelesítés

1. A felhasználó megadja a nevét és jelszavát a KDC-nek (ez történhet egy bejelentkező dialógus ablakkal vagy akár smart kártyával is).
2. A KDC kiad egy *TGT* (*Ticket Granting Ticket*=Jegymegadási Jegyet).
3. Ezzel a *TGT*-vel lehet elérni a *TGS*-t (*Ticket Granting Service*=Jegymegadási Szolgáltatás).
4. A *TGS* kiad az ügyfélnek egy szolgáltatásjegyet.
5. A hálózati szolgáltatások eléréséhez az ügyfél ezt a szolgáltatás jegyet fogja bemutatni. Ezt nevezik kölcsönös hitelesítésnek, mert a jegy hitelesíti az ügyfelet a szolgáltatásnak, a szolgáltatás pedig hitelesíti magát az ügyfélnek.

Az adatok sérthetetlensége

A szigorú hitelesítési eljárás keveset ér, ha a kiszolgáló és az ügyfél között módosulhat az információ. Harmadik fél, aki az adatokat megfejteni nem tudja, esetleg módosíthatja, ezzel értéktelenné téve azokat. Ez ellen úgy védekezik a Kerberos, hogy minden csomaghoz képez egy ellenőrző összeget (*checksum*). Bármilyen adatváltozás történik, az ellenőrző összeg alapján azonnal kiderül. A biztonság fokozása érdekében az adatcsomagok és az ellenőrző összegek kódolva, egymástól függetlenül, külön kerülnek átvitelre.

Delegálás

A delegálás a hitelesítési jog átruházása egy közbenső személy vagy számítógépfiók számára:

- Az ügyfél nem a KDC-hez, hanem a köztes állomáshoz fordul TGT kéréssel.
- Az állomás továbbítja a kérést a KDC-hez, a kommunikáció ezután köztük zajlik.
- Az állomás az ügyfél nevében hozzájut a szolgáltatásjegyhez.
- Ezzel a jeggyel pedig a szolgáltatást nyújtó kiszolgálóhoz fordul.
- Megtörténik a hitelesítés.
- Ezen a ponton megszűnik a köztes állomás szerepe, ettől kezdve a kommunikáció már az ügyfél és a szolgáltatás közt zajlik.

Alapértelmezésben delegálási joggal csak a tartománygazdák rendelkeznek.

A Kerberos és az NTLM összehasonlítása

A Windows 2000 előtti operációs rendszerek bevált hitelesítési eljárása az NTLM volt, de azóta a Kerberos lett, aminek a fontosabb okai a következőek:

- A Kerberos oda - vissza alapú hitelesítést tesz lehetővé az ügyfél és a szolgáltatás között.
- Delegálható a hitelesítés.
- Gyorsabb.
- Nem kell a szolgáltatás kiszolgálójának minden alkalommal a tartományvezérlőhöz fordulnia az ügyfél hitelesítése érdekében.
- Más platformokon (pl. Unix) is használható, nem csak a Windows alapú rendszerekben.

Gyakorlati tudnivalók

A Kerberos az alapértelmezett hitelesítési protokoll a Windows 2000-től. így csak akkor nincs használatban, ha Windows 2000 előtti rendszerek hitelesítésére van szükség, ekkor az NTLM alkalmazására kerül sor. A csoportházirendben vannak előírva a Kerberos irányelvek. Indítsuk el a Felügyeleti eszközök→Tartományi biztonsági házirend és/vagy a Tartományvezérlő biztonsági házirend MMC konzolokat. Tallózzunk el a Biztonsági beállítások→Fiókházirend→Kerberos irányelv beállításokhoz. Az alábbi állítási lehetőségeink vannak:

- Felhasználói bejelentkezési korlátozások érvényesítése. Engedélyezése esetén minden felhasználói kérés a KDC-n keresztül kerül érvényesítésre.
- Felhasználói jegy maximális élettartama. Ez a TGT (jegymegadási jegy) felhasználhatóságának élettartama. Alapértelmezett érték: 10 óra.
- Felhasználói jegy megújításának maximális élettartama. Maximum ennyi ideig lehet megújítani a szolgáltatás jegyet, mielőtt újat kell igényelni. Alapértelmezett érték: 7 nap.
- Számítógép időszinkronjának maximális tűrése. Maximum ennyi eltérés lehet a kliens és a kiszolgáló órái között. Ha ennél nagyobb, a kliens órája igazodik a kiszolgálóéhoz. Ezzel kiszűrhetők az ismétlődő támadások (replay attacks), illetve az útközben elfogott és esetleg sikeresen módosított csomagok, amikor visszakerülnek a hálózatra érvénytelenné válnak. A Kerberos protokoll időbélyegeket épít be a hálózatra küldött csomagokba, ezért fontos, hogy ne legyen nagy különbség a számítógépek rendszerideje között.
- Szolgáltatásjegy maximális élettartama. A TGS által kiadott jegyek (amellyel elérhetők a szolgáltatások) ennyi ideig használhatók fel. Ez folyamatos szolgáltatás használatot jelenti, lejárta után meg kell újítani. Alapértelmezett érték: 600 perc.

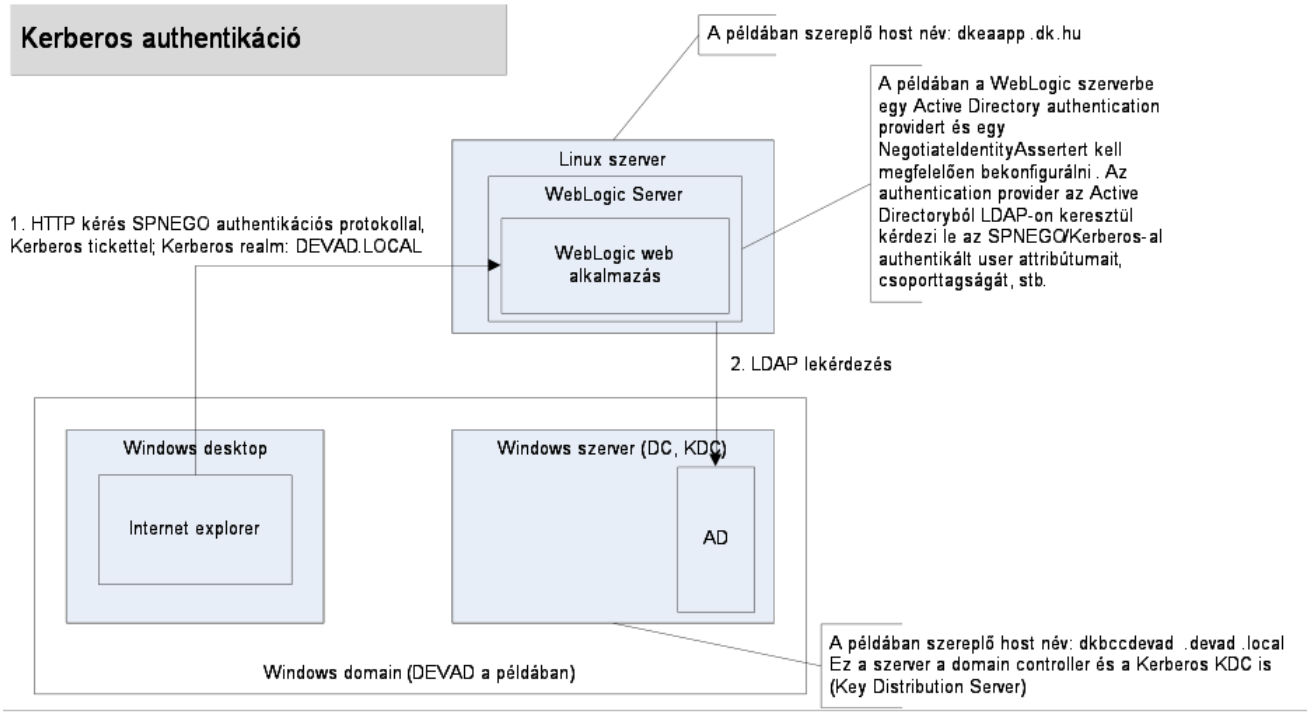
Tartományvezérlőkön folyamatosan kell futnia a KDC-nek. Ez szolgáltatás formában van jelen és a Felügyeleti eszközök→Szolgáltatások között a "Kerberos kulcs elosztó központ" nevű automatikusan induló alkalmazást kell keresnünk.

A bemutatandó konkrét feladat

Célunk, hogy bemutassuk a Windows alapú Kerberos autentikáció használatát egy Java szerver környezetben, esetünkben ez most Oracle Weblogicra telepített web alkalmazás Windows integrált SSO használatának ismertetése lesz. A 4.2. ábra foglalja össze a megoldandó feladatot, ahol jelöltük a példa konfigurációkban alkalmazott konkrét elnevezéseket (host név, stb.) is. Ezeket az elnevezéseket természetesen az aktuális telepítési környezetben érvényes értékekkel kell helyettesíteni. A Windows domainben lévő desktop az Internet Explorer böngésző segítségével éri el egy Weblogicba telepített web alkalmazást. A desktop gép egy Windows domainben van (DEVAD), a web alkalmazást pedig úgy konfiguráltuk, hogy *SPNEGO* autentikációt használjon. Windows domainben a SPNEGO[2] két konkrét autentikációs algoritmust használhat: *NTLM* vagy *Kerberos*. Jelen dokumentum azzal foglalkozik, hogy tudunk Kerberos autentikációt használni SPNEGO-val. Amennyiben a környezetet helyesen konfiguráljuk, az Internet Explorer elküldi a Kerberos ticketet a Weblogic alkalmazásnak, ami validálja azt a birtokában lévő kulcs segítségével. Amennyiben a ticket helyes, a Weblogic szerver a ticketből megállapítja a desktopon a domainbe bejelentkezett

[2]SPNEGO=Simple and Protected GSSAPI Negotiation Mechanism

user azonosítóját, és LDAP-on keresztül lekérdezi az AD-ból a user csoportjait, ezáltal létrejön a Weblogicban egy JAAS módon autentikációt subject. Ezt a subjectet a Weblogicba telepített alkalmazás már úgy használhatja, mint egy normál JAAS security context. Az ábrán a Kerberos protokollon történő kommunikációt nem tüntettük fel. A következő fejezetek azt írják le, hogy a fenti módon működő autentikációt hogyan lehet bekonfigurálni. A Weblogicban futó minta web alkalmazás URL-je: `http://dkeaapp.dk.hu:15301/ssologin`.



4.2. ábra. A Kerberos hitelesítés használata Weblogicban

Windows Server oldali konfiguráció

Az encryption type beállítás

A Windows Serveren engedélyezni kell minden *encryption type*-ot a Kerberoshoz. Ehhez használni kell a *Group Policy Management Console* programot (command line-ból: *gpmc.msc*). A 4.3. ábra mutatja a beállítás módját, szürkével kiemelve látszanak a kiválasztandó menüpontok. A feljövő dialógus ablak *Security Policy Setting* fülén ki kell választani az összes elérhető encryption type-ot. A beállítás után a Windows Server újraindítása szükséges.

Active Directory (AD)

Az Active Directory-ban az alábbi konfigurációk elvégzésére van szükség.

Egy technikai user létrehozása

Szükséges egy technikai felhasználó létrehozása, amit hozzárendelünk a Weblogic-ban futó szolgáltatáshoz a Weblogic host neve alapján. Ez azért szükséges, mert ehhez a felhasználóhoz generáljuk le azt a kulcsot, amivel az adott Weblogic alkalmazás autentikációjához szükséges Kerberos ticketek titkosítva és hitelesítve lesz-

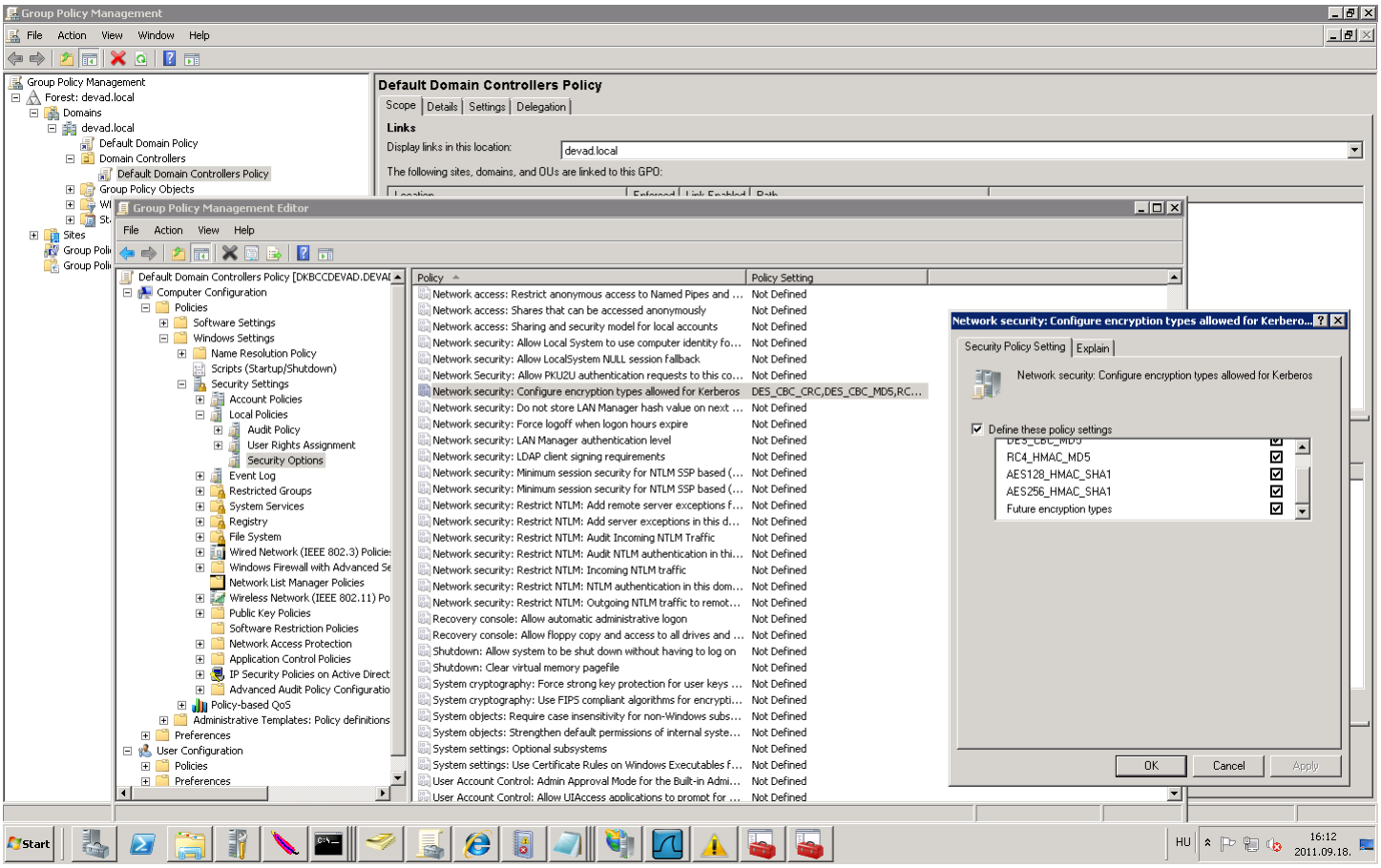
nek. A kulcsot csak a Windows Server (KDC) és a Weblogic alkalmazás fogja ismerni. Amikor az Internet Explorerben megnyitjuk a `http://dkeaapp.dk.hu:15301/ssologin` URL-t, azaz hozzáférést kezdeményezünk a Weblogic alkalmazáshoz, az IE Kerberos protokollon lekér egy titkosító kulcsot a KDC-től (Windows Server) az URL-ből képzett SPN (*Service Principal Name*) alapján. Az SPN formátuma:

```
HTTP/<teljes host név>@<Kerberos realm név, ➥
    nagybetűkkel>
```

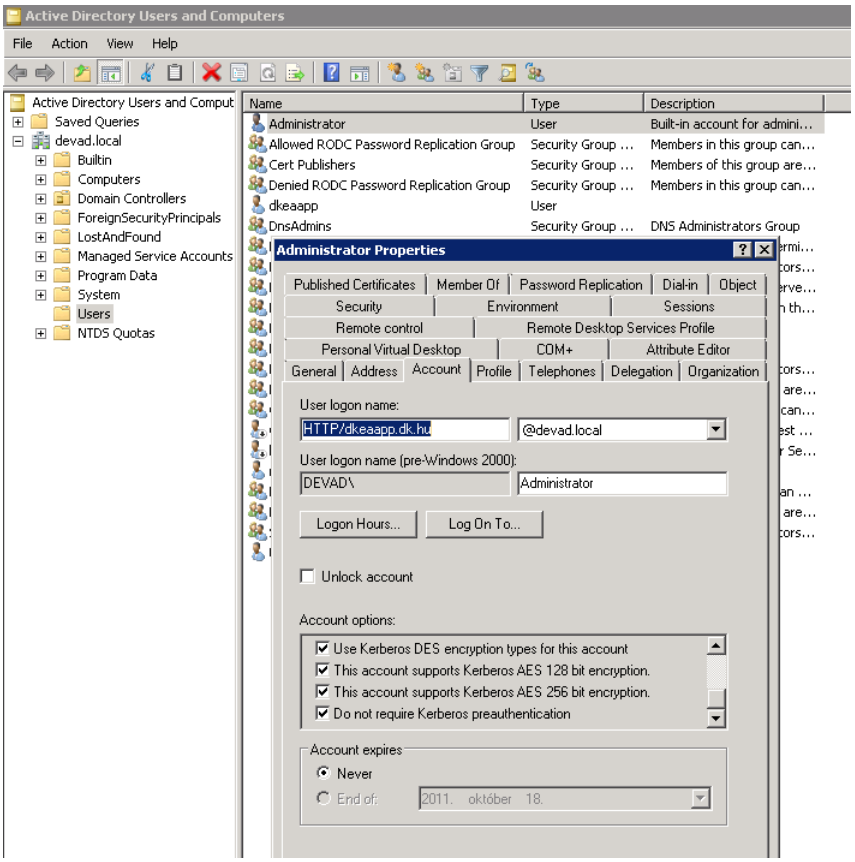
A mi esetünkben:

```
http/dkeaapp.dk.hu@DEVAD.LOCAL
```

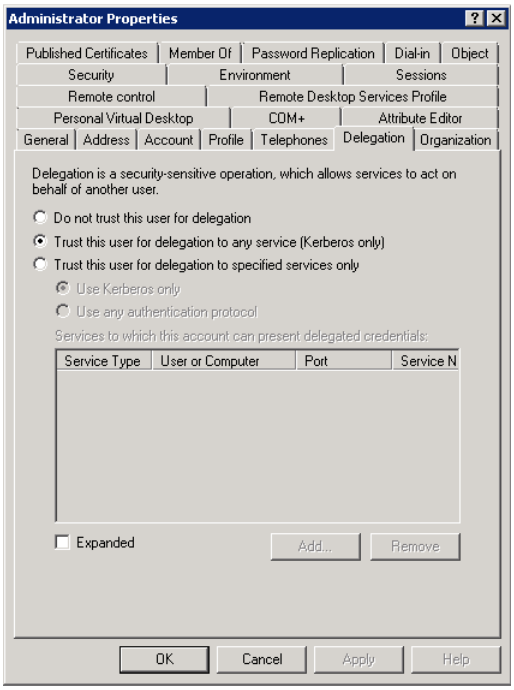
A titkosító kulcsot a KDC az SPN-hez rendelt technikai felhasználó jelszavából képzi, tehát a desktop géppel a kulcsokat ténylegesen nem osztja meg. Az Internet Explorer ezzel a titkosító kulccsal készít egy Kerberos tokent, amit a Weblogicnak elküld a `http://dkeaapp.dk.hu:15301/ssologin` kérés HTTP headerében. A Weblogic alkalmazás a nála lévő, technikai felhasználóhoz tartozó 1. kulcs segítségével validálja le a tokent, és fejti belőle vissza a desktopon a Windows domainbe bejelentkezett user nevét.



4.3. ábra. Group Policy Management Console



4.4. ábra. Active Directory Users and Computers alkalmazás



4.5. ábra. A delegáció beállítása

A technikai user létrehozásának menete:

1. Hozzuk létre a usert (Jelen példában az *Administrator* usert használjuk erre a célra). Fontos megjegyezni, hogy mivel a user jelszavát nem kell megosztani a Weblogic oldallal sem, ezért a megoldás biztonságosnak tekinthető.

2. Hozzuk létre az SPN-t a Windows gépen a *setspn* utility segítségével, amihez a következő parancsot kell lefuttatni:

```
setspn -A <SPN neve a Kerberos realm név➥
    nélkül> <technikai user neve>
```

Példánkban ez:

```
setspn -A HTTP/dkeaapp.dk.hu ➥
    Administrator
```

Érdemes létrehozni egy olyan SPN-t is, amiben csak a host név szerepel, nem a host teljes neve. Példánkban ez ezt jelenti:

```
setspn –A HTTP/dkeaapp Administrator
```

3. Hozzuk létre a technikai userhez tartozó kulcsot, amit a Weblogic alkalmazás alá másolunk majd. Ehhez a *ktpass* utility-t kell lefuttatni a Windows Serveren a következőképpen:

```
ktpass –out <host név>.host.keytab –➥
    mapuser <technikai user neve>@<➥
    Kerberos realm név> –princ <SPN ➥
    teljes neve> –pass <a technikai user➥
     jelszava> –ptype KRB5_NT_PRINCIPAL
```

Példánkban ez így néz ki:

```
ktpass –out dkeaapp.host.keytab –mapuser➥
    Administrator@DEVAD.LOCAL –princ ➥
    HTTP/dkeaapp.dk.hu@DEVAD.LOCAL –pass➥
    P@ssword –ptype KRB5_NT_PRINCIPAL
```

A parancs lefuttatása után létrejön egy *.keytab* fájl, ami esetünkben *dkeaapp.host.keytab* és őt a Weblogic-ot futtató szerverre kell majd másolni.

4. Állítsuk be a technikai userre a következőket az *Active Directory Users and Computers* alkalmazás segítségével (4.4. ábra, *Account* fül). Érdemes beállítani azt is itt, hogy a user jelszava soha ne járjon le (Password never expires). Ugyanezen ablak *Delegation* fülén válasszuk ki a 4.5. ábrán látható középső rádió gombot.

5. Reseteljük a technikai user jelszavát, ami egy biztonságnövelő lépés csupán.

Megjegyzés: Fontos, hogy az előző pontokban létrehozott SPN-ekhez ne legyen több user bekonfigurálva, mert ez a legkülönfélébb hibákat okozhatja. Ezt a legkönnyebben úgy ellenőrizhetjük, hogy egy szöveges fájlba dumpoljuk az AD tartalmát (*ldifde –f <fájl neve>* parancs), és megnézzük, hogy az adott SPN hányszor szerepel az AD-ban. Ha többször, akkor a nem megfelelő bejegyzéseket a *setspn –D <SPN> <user>* paranccsal törölhetjük.

Egy normál domain user létrehozása

Hozzunk létre egy normál domain usert, amivel az SSO-t fogjuk tesztelni a desktopon (ez esetünkben a *dkeaapp* nevű user lesz). Ennek a usernek az megadott beállításokat kell megadni az ismertetett *Account* fülön.

Egy LDAP lekérdező user létrehozása

Hozzunk létre egy usert, amit arra használunk, hogy a nevében a Weblogic LDAP lekérdezéseket tudjon futtatni.

Windows kliens oldali konfiguráció

A Windows desktopon az Internet Explorert úgy kell beállítani, hogy az adott URL-re használja a *Windows Integrated Authentication*t (IWA). Ehhez az Internet Explorerben a Tools/Internet Settings menübe lépve a Security fülön fel kell venni a Local Intranet helyei közé a Weblogic hostot (esetünkben: `http://dkeaapp.dk.hu`), és az Advanced fülön engedélyezni kell az Integrated Windows Authenticationt.

Weblogic oldali konfiguráció

A következőkben az Oracle Weblogic szerverben elvégzendő beállításokat ismertetjük.

Active Directory Authentication Provider

Az első lépés az *Active Directory Authentication Provider* bekonfigurálása, amit a Weblogic Admin Console Security → Realm → Providers → Authentication → New Active Directory kiválasztásával tehetünk meg. Példánkban a következő paramétereket használtuk:

```
name: ActiveDirectory
control-flag: OPTIONAL
host: dkbcc.devad.local
user-name-attribute: sAMAccountName
principal: CN=Administrator,CN=Users,DC=devad,DC=local
user-base-dn: dc=devad,dc=local
credential-encrypted: P@ssword
user-from-name-filter: (&(sAMAccountName=%u)(objectclass=user))
group-base-dn: dc=devad,dc=local
use-retrieved-user-name-as-principal: true
```

Ez a hitelesítő szolgáltató (provider) biztosítja, hogy a Weblogic képes legyen a bekonfigurált AD-t felhasználókat hitelesítő adatbázisként használni.

SPNEGO Identity Asserter

SPNEGO Identity Asserter konfigurálása. Ehhez egyszerűen alap beállításokkal létre kell hozni egy új *NegotiateIdentityAsserter*-t (Security → Realm → Providers → Authentication → New NegotiateIdentityAsserter). Ügyeljünk, hogy az asserter mindkét token típusa aktív legyen, bár alapértelmezetten az asserter így jön létre.

A SPNEGO védelme alatt álló erőforrás esetén a szerver a HTTP fejlécben ezt kéri a kliens-től.

```
WWW-Authenticate: Negotiate
```

A helyes sorrend beállítása

Adjuk meg a helyes sorrendet az authentikátoroknak. Legyen az első az SPNEGO-ra létrehozott asserter, a második az AD-hoz létrehozott authentikátor, majd tegyünk minden egyéb authentikátort opcionálisra (control flag: *OPTIONAL*).

A Kerberos bekonfigurálása

Hozzuk létre a Weblogic Server domain könyvtárában egy *spnego_conf* nevű könyvtárat, és helyezzük el benne a következő fájlokat:

1. Az előzőleg létrehozott keytab fájlt, ami példánkban most a *dkeaapp.host.keytab*.
2. Egy *login.conf* nevű fájlt a következő tartalommal:

```
com.sun.security.jgss.krb5.initiate
{
        com.sun.security.auth.module.Krb5LoginModule required
        storeKey=true
       debug=true
       useKeyTab=true
       keyTab="<keytab fájl helye>"
       useTicketCache=true
       principal="<SPN neve>";
};

com.sun.security.jgss.krb5.accept
{
```

```
        com.sun.security.auth.module.Krb5LoginModule required
        storeKey=true
       debug=true
       useKeyTab=true
       keyTab="<keytab fájl helye>"
       useTicketCache=true
       principal="<SPN neve>";
};
```

Példánkban ezt a fájlt így kell konkrét értékekkel felruházni:

```
com.sun.security.jgss.krb5.initiate
{
        com.sun.security.auth.module.Krb5LoginModule required
        storeKey=true
       debug=true
       useKeyTab=true
       keyTab="/home/wlsapp/ad_domain/spnego_conf/dkeaapp.host.keytab"
       useTicketCache=true
       principal="HTTP/dkeaapp.dk.hu@DEVAD.LOCAL";
};

com.sun.security.jgss.krb5.accept
{
        com.sun.security.auth.module.Krb5LoginModule required
        storeKey=true
       debug=true
       useKeyTab=true
       keyTab="/home/wlsapp/ad_domain/spnego_conf/dkeaapp.host.keytab"
       useTicketCache=true
       principal="HTTP/dkeaapp.dk.hu@DEVAD.LOCAL";
};
```

A *debug=true* értéket éles környezetben érdemes *debug=false*-ra változtatni. A Kerberos realm nevét nagybetűvel kell írni (a példában ez *DEVAD.LOCAL*)

3. Egy *krb5.conf* fájlt a következő tartalommal:

```
[libdefaults]
        default_realm = <Kerberos realm neve nagybetűvel>
        default_tkt_enctypes = rc4-hmac aes256-cts des-cbc-md5
       default_tgs_enctypes = rc4-hmac aes256-cts des-cbc-md5
        permitted_enctypes   = aes256-cts aes128-cts rc4-hmac des3-cbc-sha1 ➥
            des-cbc-md5 des-cbc-crc
[realms]
        <Kerberos realm neve nagybetűvel>  = {
                kdc = <KDC host neve>
                default_domain = <Windows DNS domain név>
}

[domain_realm]
.<Windows DNS domain név> = <Kerberos realm neve nagybetűvel>
```

Esetünkben ez a fájl így néz ki:

```
[libdefaults]
        default_realm = DEVAD.LOCAL
        default_tkt_enctypes = rc4-hmac aes256-cts des-cbc-md5
       default_tgs_enctypes = rc4-hmac aes256-cts des-cbc-md5
        permitted_enctypes   = aes256-cts aes128-cts rc4-hmac des3-cbc-sha1 ➥
            des-cbc-md5 des-cbc-crc
[realms]
        DEVAD.LOCAL  = {
                kdc = dkbccdevad.devad.local
                default_domain = devad.local
}

[domain_realm]
        .devad.local = DEVAD.LOCAL
```

Java system property beállítás

A Weblogic indító szkriptjében adjuk meg a következő Java system property-ket:

```
-Djava.security.krb5.conf=<krb5.conf path>spnego_conf/➥
    krb5.conf
-Djava.security.auth.login.config=<login.conf path>➥
    spnego_conf/login.conf
-Djavax.security.auth.useSubjectCredsOnly=false
-Dweblogic.security.enableNegotiate=true
```

Példánkban:

```
-Djava.security.krb5.conf=spnego_conf/krb5.conf
-Djava.security.auth.login.config=spnego_conf/login.➥
    conf
-Djavax.security.auth.useSubjectCredsOnly=false
-Dweblogic.security.enableNegotiate=true
```

Amennyiben hibakereséshez debugolni szeretnénk:

```
-Dweblogic.Stdout.DebugEnabled=true
-Dweblogic.StdoutSeverityLevel=64
-Dsun.security.krb5.debug=true
-Dsun.security.jgss.debug=true
```

A beállítások után indítsuk újra a Weblogic szervert!

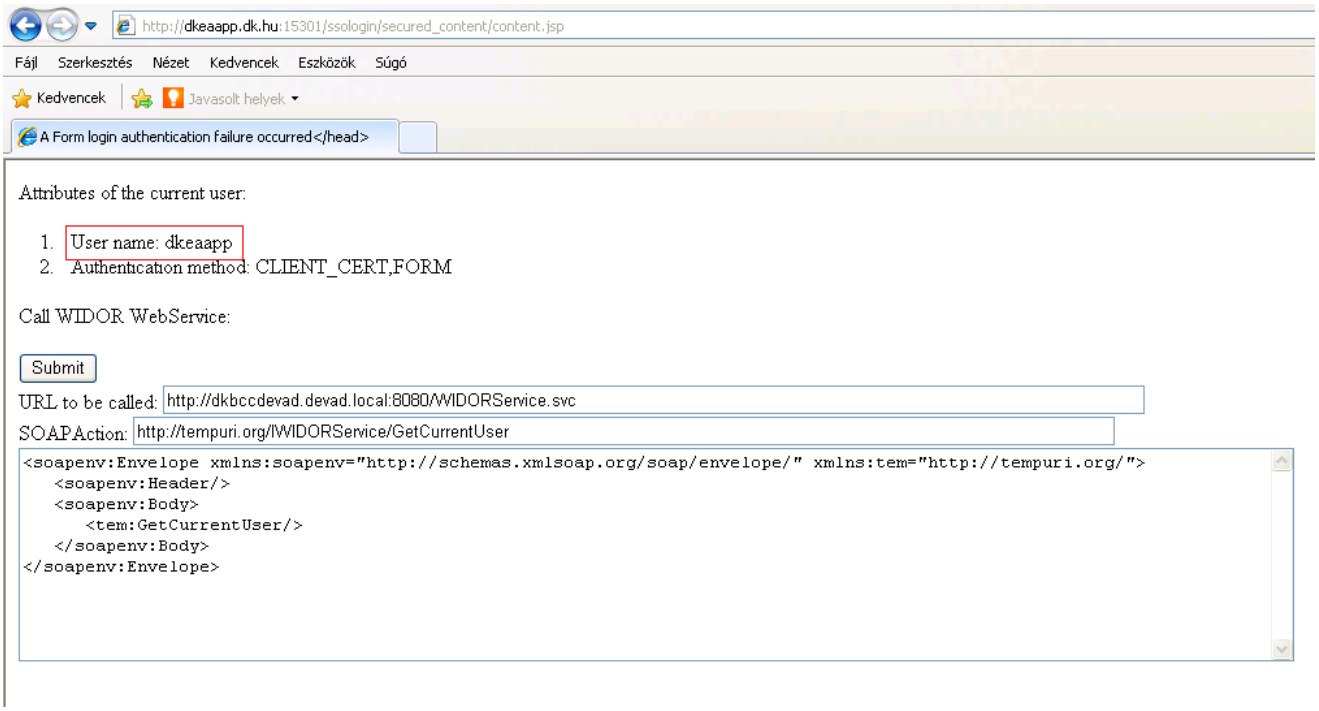
A minta webalkalmazás

A Kerberos SSO használatához úgy kell elkészíteni a web alkalmazást, hogy használja a HTTP kommunikációban szereplő Kerberos ticketeket. Ehhez a *web.xml* fájlt kell megfelelő módon előállítani, más különösebb teendő nincsen (4-1. programlista). Magát a fájlt ugyanúgy kell megadni, mint minden más normál *JAAS*-t használó web alkalmazásnál. A 27. sorban megadott autentikáció módja tehát mindenképpen *CLIENT-CERT* kell, hogy legyen, de opcionálisan lehet alkalmazni fallback mechanizmust. Ez a példában egy form based autentikáció, aminek a konfigurációját a 28-31 sorok között határoztuk meg. Ehhez kellett definiálni a *form-login-config* részt a *login* és *error* oldalak megadásával. Erre akkor lehet szükség, ha az alkalmazást olyan desktopról is el lehet érni, ami nincs benne a Windows domainben, és ahol a desktop nem küld SPNEGO tokeneket. Ilyenkor explicit szükséges bekérni a felhasználótól a user nevet és a jelszót a megadott login oldal felmutatásával. Természetesen továbbra is a Windows domainben érvényes user nevek és jelszavak használatosak, hiszen ekkor is az AD authentikátor végzi az autentikációt (nem csak a csoporttagság lekérdezését). A példához készített mintaalkalmazás az *ssologin.war* fájlban található. Ezt lehet telepíteni a Weblogic szerverbe. Ügyelni kell arra, hogy a *weblogic.xml* fájlban (4-2. programlista) lévő *principal-name* tag-ek között szereplő user neveket/user csoportokat a helyi környezethez igazítsuk. Ezek a userek, illetve cso-

portok az Active Directory-ban kell, hogy létezzenek. Amennyiben beállítottuk az összes, előző pontokban leírt konfigurációt és telepítettük a Weblogic-on az *ssologin.war* alkalmazást, a DEVAD domainben lévő Windows desktop gépen a *dkeaapp* userrel bejelentkezve indítsuk el az Internet Explorert a következő URL-el: `http://dkeaapp.dk.hu:15301/ssologin`. Amennyiben minden rendben van, a 4.6. ábrán mutatott tartalmat kell látnunk. Az alkalmazás visszaírja a képernyőre a domainbe bejelentkezett felhasználó nevét, ezzel bizonyítva, hogy a Kerberos autentikáció sikerült a Weblogic oldalon. Ezt a működést a 4-3. programlistán tanulmányozható *content.jsp* lap biztosítja, ami a hitelesítést követően hajtódhat csak végre. Ez a JSP lap használja a 4-4-től a 4-8. programlistán szereplő Java osztályokat, ami a példa teljes közlése érdekében változatlan formában helyeztünk el. Amennyiben mégis FORM alapú hitelesítésre irányít minket a szerver, úgy a 4-9. és 4.10. programlista mutatja az ehhez tartozó login és error oldalt. Esetleges hibák:

- Amennyiben a következő hiba jön a Weblogic oldalon: „KDC has no support for encryption type (14)", valószínűleg a kliensről érkező token nem olyan kódolással érkezik, amilyen kódoláshoz a keytab fájlban kulcs található. Ilyenkor célszerű megnézni wiresharkkal a http forgalmat, megnézni, hogy milyen kódolással érkezik a Weblogicba a token (a wireshark ezt megmutatja, ismeri a KRB5 protokollt), és ezzel újragenerálni a keytab fájlt (ilyenkor a *ktpass* parancsnak *–crypto* kapcsolóval meg kell adni ugyanazt a kódolási típust, amit a wiresharkban látunk)

- Ügyelni kell arra, hogy a 3 kommunikációban szereplő gép (kliens, Windows Server, Weblogic) órája szinkronban legyen!



4.6. ábra. A minta SSO webalkalmazás

```
// 4-1. programlista: web.xml

<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4" xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>ssologin</display-name>

    <welcome-file-list>
        <welcome-file>secured_content/content.jsp</welcome-file>
    </welcome-file-list>

    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Content</web-resource-name>
            <url-pattern>/secured_content/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>POST</http-method>
        </web-resource-collection>
        <auth-constraint>
            <role-name>*</role-name>
        </auth-constraint>
        <user-data-constraint>
            <transport-guarantee>NONE</transport-guarantee>
        </user-data-constraint>
    </security-constraint>

    <login-config>
        <auth-method>CLIENT-CERT,FORM</auth-method>
        <form-login-config>
            <form-login-page>/login.jsp</form-login-page>
            <form-error-page>/error.jsp</form-error-page>
        </form-login-config>
    </login-config>

    <security-role>
        <role-name>sso_users_or_groups</role-name>
    </security-role>

</web-app>
```

```
// 4-2. programlista: weblogic.xml

<?xml version="1.0" encoding="UTF-8"?>
<wls:weblogic-web-app xmlns:wls="http://xmlns.oracle.com/weblogic/weblogic-web-app" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd http://xmlns.oracle.com/weblogic/weblogic-web-app http://xmlns.oracle.com/weblogic/weblogic-web-app/1.0/weblogic-web-app.xsd">
    <wls:weblogic-version>10.3.2</wls:weblogic-version>
    <wls:context-root>ssologin</wls:context-root>

    <wls:security-role-assignment>
        <wls:role-name>sso_users_or_groups</wls:role-name>
        <!-- An AD group name should come here -->
        <wls:principal-name>techiccm</wls:principal-name>
    </wls:security-role-assignment>

    <wls:charset-params>
        <wls:input-charset>
            <wls:resource-path>/*</wls:resource-path>
            <wls:java-charset-name>UTF-8</wls:java-charset-name>
        </wls:input-charset>
    </wls:charset-params>
```

```
20
21      <wls:session-descriptor>
22          <wls:persistent-store-type>async-replicated-if-clustered</wls:persistent-store-type>
23      </wls:session-descriptor>
24
25      <wls:jsp-descriptor>
26          <wls:keepgenerated>true</wls:keepgenerated>
27          <wls:page-check-seconds>-1</wls:page-check-seconds>
28          <wls:precompile>false</wls:precompile>
29          <wls:encoding>UTF-8</wls:encoding>
30      </wls:jsp-descriptor>
31
32  </wls:weblogic-web-app>
```

Ez a lap fog megjelenni hitelesítés után:

```
1   // 4-3. programlista: content.jsp
2
3   <!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
4   <html>
5   <head><title>A Form login authentication failure occurred</head></title>
6   <body>
7   <P>Attributes of the current user:
8   <OL>
9   <LI>User name: <%=request.getUserPrincipal()%>
10  <LI>Authentication method: <%=request.getAuthType()%>
11  </OL>
12  </P>
13
14  <%
15
16  if(request.getParameter("url") != null && request.getParameter("soap") != null) {
17
18  try {
19    hu.alerant.spnego.SpnegoSOAPConnection soapConnection = new hu.alerant.spnego.➥
        SpnegoSOAPConnection();
20    javax.xml.soap.MessageFactory msgFactory = javax.xml.soap.MessageFactory.newInstance();
21    javax.xml.soap.MimeHeaders mimeHeaders = new javax.xml.soap.MimeHeaders();
22    mimeHeaders.addHeader("Content-Type","text/xml; charset=UTF-8");
23    java.io.ByteArrayInputStream stream = new java.io.ByteArrayInputStream(request.getParameter("➥
        soap").getBytes("UTF-8"));
24    javax.xml.soap.SOAPMessage soapRequest = msgFactory.createMessage(mimeHeaders, stream);
25    javax.xml.soap.SOAPMessage soapResponse = soapConnection.call(soapRequest, request.➥
        getParameter("url"));
26    java.io.ByteArrayOutputStream bos = new java.io.ByteArrayOutputStream();
27    soapResponse.writeTo(bos);
28  %>
29  <p>
30  Response from WIDOR: <br/>
31  <pre>
32  <%=new String(bos.toByteArray())%>
33  </pre>
34  </p>
35  <%
36
37  }
38  catch(Exception e) {
39  %>
40  <p>
41  Error occured while accessing WIDOR: <br/>
42  <pre>
43  <%=e.getMessage()%>. See the server log files for detailes!
44  </pre>
45  </p>
46  <%
```

```
47
48    e.printStackTrace();
49  }
50
51  }
52
53  %>
54
55  <p>Call WIDOR WebService:
56
57  <form method="POST" action="/ssologin/secured_content/content.jsp">
58
59  <input type="submit" value"Submit">
60
61  <br/>
62
63  URL to be called: <input type="text" size="120" name="url" value='<%=request.getParameter("url")➥
        %>'/>
64
65  <br/>
66
67  <textarea rows="100" cols="120" name="soap">
68  <%=request.getParameter("soap")%>
69  </textarea>
70
71  </form>
72
73  </p>
74
75
76  </body>
77  </html>
```

```
1   // 4-4. programlista:
2
3   package hu.alerant.spnego;
4
5   public final class Base64 {
6
7       private static final String ALPHABET =
8               "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/";
9
10      private Base64() {
11          // default private
12      }
13
14      public static String encode(final byte[] bytes) {
15          int length = bytes.length;
16
17          if (length == 0) {
18              return "";
19          }
20
21          final StringBuilder buffer =
22                  new StringBuilder((int) Math.ceil(length / 3d) * 4);
23          final int remainder = length % 3;
24          length -= remainder;
25          int block;
26          int idx = 0;
27          while (idx < length) {
28              block = ((bytes[idx++] & 0xff) << 16) | ((bytes[idx++] & 0xff) << 8)
29              | (bytes[idx++] & 0xff);
30              buffer.append(ALPHABET.charAt(block >>> 18));
31              buffer.append(ALPHABET.charAt((block >>> 12) & 0x3f));
32              buffer.append(ALPHABET.charAt((block >>> 6) & 0x3f));
```

```
33              buffer.append(ALPHABET.charAt(block & 0x3f));
34          }
35          if (remainder == 0) {
36              return buffer.toString();
37          }
38          if (remainder == 1) {
39              block = (bytes[idx] & 0xff) << 4;
40              buffer.append(ALPHABET.charAt(block >>> 6));
41              buffer.append(ALPHABET.charAt(block & 0x3f));
42              buffer.append("==");
43              return buffer.toString();
44          }
45          block = (((bytes[idx++] & 0xff) << 8) | ((bytes[idx]) & 0xff)) << 2;
46          buffer.append(ALPHABET.charAt(block >>> 12));
47          buffer.append(ALPHABET.charAt((block >>> 6) & 0x3f));
48          buffer.append(ALPHABET.charAt(block & 0x3f));
49          buffer.append("=");
50          return buffer.toString();
51      }
52
53      public static byte[] decode(final String string) {
54          final int length = string.length();
55          if (length == 0) {
56              return new byte[0];
57          }
58
59          final int pad = (string.charAt(length - 2) == '=') ? 2
60                  : (string.charAt(length - 1) == '=') ? 1 : 0;
61          final int size = length * 3 / 4 - pad;
62          byte[] buffer = new byte[size];
63          int block;
64          int idx = 0;
65          int index = 0;
66          while (idx < length) {
67              block = (ALPHABET.indexOf(string.charAt(idx++)) & 0xff) << 18
68              | (ALPHABET.indexOf(string.charAt(idx++)) & 0xff) << 12
69              | (ALPHABET.indexOf(string.charAt(idx++)) & 0xff) << 6
70              | (ALPHABET.indexOf(string.charAt(idx++)) & 0xff);
71              buffer[index++] = (byte) (block >>> 16);
72              if (index < size) {
73                  buffer[index++] = (byte) ((block >>> 8) & 0xff);
74              }
75              if (index < size) {
76                  buffer[index++] = (byte) (block & 0xff);
77              }
78          }
79          return buffer;
80      }
81  }
```

```
1   // 4-5. programlista:
2
3   package hu.alerant.spnego;
4
5   public class Constants {
6
7       private Constants() {
8           // default private
9       }
10
11      public static final String BASIC_HEADER = "Basic";
12
13      public static final String NEGOTIATE_HEADER = "Negotiate";
14
15      public static final String AUTHN_HEADER = "WWW-Authenticate";
```

```
16
17      public static final String AUTHZ_HEADER = "Authorization";
18
19  }
```

```
1   // 4-6. programlista:
2   package hu.alerant.spnego;
3
4   final class SpnegoAuthScheme {
5
6       private static final transient byte[] EMPTY_BYTE_ARRAY = new byte[0];
7
8       private final transient String scheme;
9
10      private final transient String token;
11
12      private final transient boolean basicScheme;
13
14      private final transient boolean negotiateScheme;
15
16      public SpnegoAuthScheme(final String authScheme, final String authToken) {
17          this.scheme = authScheme;
18          this.token = authToken;
19
20          this.negotiateScheme = Constants.NEGOTIATE_HEADER.equalsIgnoreCase(authScheme);
21          this.basicScheme = Constants.BASIC_HEADER.equalsIgnoreCase(authScheme);
22      }
23
24      boolean isBasicScheme() {
25          return this.basicScheme;
26      }
27
28      boolean isNegotiateScheme() {
29          return this.negotiateScheme;
30      }
31
32      public String getScheme() {
33          return this.scheme;
34      }
35
36      public byte[] getToken() {
37          return (null == this.token) ? EMPTY_BYTE_ARRAY : Base64.decode(this.token);
38      }
39  }
```

```
1   // 4-7. programlista:
2
3   package hu.alerant.spnego;
4
5   import java.io.ByteArrayOutputStream;
6   import java.io.IOException;
7   import java.io.InputStream;
8   import java.io.OutputStream;
9   import java.net.HttpURLConnection;
10  import java.net.URL;
11  import java.security.PrivilegedActionException;
12  import java.util.Arrays;
13  import java.util.LinkedHashMap;
14  import java.util.List;
15  import java.util.Map;
16  import java.util.Set;
17  import java.util.concurrent.locks.Lock;
18  import java.util.concurrent.locks.ReentrantLock;
19  import java.util.Iterator;
```

```
20 import org.ietf.jgss.GSSContext;
21 import org.ietf.jgss.GSSCredential;
22 import org.ietf.jgss.GSSException;
23 import org.ietf.jgss.GSSManager;
24 import org.ietf.jgss.GSSName;
25 import org.ietf.jgss.Oid;
26
27 import weblogic.security.Security;
28 import com.bea.security.utils.negotiate.CredentialObject;
29 import sun.security.jgss.GSSCredentialImpl;
30 import sun.security.jgss.GSSManagerImpl;
31 import sun.security.jgss.spi.GSSCredentialSpi;
32
33 public final class SpnegoHttpURLConnection {
34
35     private static final Lock LOCK = new ReentrantLock();
36
37     private static final byte[] EMPTY_BYTE = new byte[0];
38
39     private static final GSSManager MANAGER = GSSManager.getInstance();
40
41     private transient boolean connected = false;
42
43     private transient String requestMethod = "GET";
44
45     private final transient Map<String, List<String>> requestProperties =
46         new LinkedHashMap<String, List<String>>();
47
48     private transient GSSCredential credential;
49
50     private transient boolean cntxtEstablished = false;
51
52     private transient HttpURLConnection conn = null;
53
54     private transient boolean reqCredDeleg = false;
55
56     public SpnegoHttpURLConnection() throws Exception {
57         if(Security.getCurrentSubject() == null) {
58             throw new Exception("No valid subject in the thread context. The user is not
                authenticated!");
59         }
60
61         System.out.println("Security.getCurrentSubject().getPrivateCredentials(): " +
62             Security.getCurrentSubject().getPrivateCredentials());
63
64         for(Iterator it = Security.getCurrentSubject().getPrivateCredentials().iterator(); it.
            hasNext();) {
65             Object o = it.next();
66             System.out.println("o.getClass().getName(): " + o.getClass().getName());
67             if(o instanceof CredentialObject) {
68               System.out.println("((CredentialObject)o).getCredential(): " + ((CredentialObject)
                    o).getCredential());
69               System.out.println("((CredentialObject)o).getDelegatedSub(): " + ((
                    CredentialObject)o).getDelegatedSub());
70               this.credential = ((CredentialObject)o).getCredential();
71               //System.out.println("((CredentialObject)o).getDelegatedSub().
                    getPrivateCredentials(): " + ((CredentialObject)o).getDelegatedSub().
                    getPrivateCredentials());
72               //System.out.println("((CredentialObject)o).getDelegatedSub().
                    getPrivateCredentials().size(): " + ((CredentialObject)o).getDelegatedSub().
                    getPrivateCredentials().size());
73               //for(Iterator it2 = ((CredentialObject)o).getDelegatedSub().getPrivateCredentials
                    ().iterator(); it2.hasNext();) {
74               //     Object o2 = it2.next();
75               //     System.out.println("o2.getClass().getName(): " + o2.getClass().getName());
```

```
76                //     this.credential = new GSSCredentialImpl((GSSManagerImpl)MANAGER, (➥
                      GSSCredentialSpi)o2);
77                //}
78              }
79          }
80          if(this.credential == null) {
81              throw new Exception("No GSSCredential found for the current subject. The subject ➥
                    might not have been authenticated with SPNEGO Kerberos.");
82          }
83      }
84
85      private void assertConnected() {
86          if (!this.connected) {
87              throw new IllegalStateException("Not connected.");
88          }
89      }
90
91      private void assertNotConnected() {
92          if (this.connected) {
93              throw new IllegalStateException("Already connected.");
94          }
95      }
96
97      public HttpURLConnection connect(final URL url)
98          throws GSSException, PrivilegedActionException, IOException {
99
100         return this.connect(url, null);
101     }
102
103     public HttpURLConnection connect(final URL url, final ByteArrayOutputStream dooutput)
104         throws GSSException, PrivilegedActionException, IOException {
105
106         assertNotConnected();
107
108         GSSContext context = null;
109
110         try {
111             byte[] data = null;
112
113             SpnegoHttpURLConnection.LOCK.lock();
114             try {
115                 // work-around to GSSContext/AD timestamp vs sequence field replay bug
116                 try { Thread.sleep(31); } catch (InterruptedException e) { assert true; }
117
118                 context = this.getGSSContext(url);
119                 context.requestMutualAuth(true);
120                 context.requestConf(true);
121                 context.requestInteg(true);
122                 context.requestReplayDet(true);
123                 context.requestSequenceDet(true);
124                 context.requestCredDeleg(true);
125
126                 data = context.initSecContext(EMPTY_BYTE, 0, 0);
127             } finally {
128                 SpnegoHttpURLConnection.LOCK.unlock();
129             }
130
131             this.conn = (HttpURLConnection) url.openConnection();
132             this.connected = true;
133
134             final Set<String> keys = this.requestProperties.keySet();
135             for (final String key : keys) {
136                 for (String value : this.requestProperties.get(key)) {
137                     this.conn.addRequestProperty(key, value);
138                 }
```

```
139             }
140
141             // TODO : re-factor to support (302) redirects
142             this.conn.setInstanceFollowRedirects(false);
143             this.conn.setRequestMethod(this.requestMethod);
144
145             this.conn.setRequestProperty(Constants.AUTHZ_HEADER
146                 , Constants.NEGOTIATE_HEADER + ' ' + Base64.encode(data));
147
148             if (null != dooutput && dooutput.size() > 0) {
149                 this.conn.setDoOutput(true);
150                 dooutput.writeTo(this.conn.getOutputStream());
151             }
152
153             this.conn.connect();
154
155             final SpnegoAuthScheme scheme = getAuthScheme(
156                     this.conn.getHeaderField(Constants.AUTHN_HEADER));
157
158             // app servers will not return a WWW-Authenticate on 302, (and 30x...?)
159             if (null == scheme) {
160                 System.out.println("getAuthScheme(...) returned null.");
161
162             } else {
163                 data = scheme.getToken();
164
165                 if (Constants.NEGOTIATE_HEADER.equalsIgnoreCase(scheme.getScheme())) {
166                     SpnegoHttpURLConnection.LOCK.lock();
167                     try {
168                         data = context.initSecContext(data, 0, data.length);
169                     } finally {
170                         SpnegoHttpURLConnection.LOCK.unlock();
171                     }
172
173                     // TODO : support context loops where i>1
174                     if (null != data) {
175                         System.out.println("Server requested context loop: " + data.length);
176                     }
177
178                 } else {
179                     throw new UnsupportedOperationException("Scheme NOT Supported: "
180                             + scheme.getScheme());
181                 }
182
183                 this.cntxtEstablished = context.isEstablished();
184             }
185         } finally {
186             this.dispose(context);
187         }
188
189         return this.conn;
190     }
191
192     private void dispose(final GSSContext context) {
193         if (null != context) {
194             try {
195                 SpnegoHttpURLConnection.LOCK.lock();
196                 try {
197                     context.dispose();
198                 } finally {
199                     SpnegoHttpURLConnection.LOCK.unlock();
200                 }
201             } catch (GSSException gsse) {
202                 System.out.println("call to dispose context failed." + gsse);
203                 gsse.printStackTrace();
```

```
204             }
205         }
206 
207         //if (null != this.credential) {
208         //    try {
209         //        this.credential.dispose();
210         //    } catch (final GSSException gsse) {
211         //        System.out.println("call to dispose credential failed." + gsse);
212         //        gsse.printStackTrace();
213         //    }
214         //}
215 
216     }
217 
218     public void disconnect() {
219         this.dispose(null);
220         this.requestProperties.clear();
221         this.connected = false;
222         if (null != this.conn) {
223             this.conn.disconnect();
224         }
225     }
226 
227     public boolean isContextEstablished() {
228         return this.cntxtEstablished;
229     }
230 
231     private void assertKeyValue(final String key, final String value) {
232         if (null == key || key.isEmpty()) {
233             throw new IllegalArgumentException("key parameter is null or empty");
234         }
235         if (null == value) {
236             throw new IllegalArgumentException("value parameter is null");
237         }
238     }
239 
240     public void addRequestProperty(final String key, final String value) {
241         assertNotConnected();
242         assertKeyValue(key, value);
243 
244         if (this.requestProperties.containsKey(key)) {
245             final List<String> val = this.requestProperties.get(key);
246             val.add(value);
247             this.requestProperties.put(key, val);
248         } else {
249             setRequestProperty(key, value);
250         }
251     }
252 
253     public void setRequestProperty(final String key, final String value) {
254         assertNotConnected();
255         assertKeyValue(key, value);
256 
257         this.requestProperties.put(key, Arrays.asList(value));
258     }
259 
260     private GSSContext getGSSContext(final URL url) throws GSSException
261         , PrivilegedActionException {
262 
263         return getGSSContext(this.credential, url);
264     }
265 
266     public InputStream getErrorStream() throws IOException {
267         assertConnected();
268 
```

```
269         return this.conn.getInputStream();
270     }
271 
272     public String getHeaderField(final int index) {
273         assertConnected();
274 
275         return this.conn.getHeaderField(index);
276     }
277 
278     public String getHeaderField(final String name) {
279         assertConnected();
280 
281         return this.conn.getHeaderField(name);
282     }
283 
284     public String getHeaderFieldKey(final int index) {
285         assertConnected();
286 
287         return this.conn.getHeaderFieldKey(index);
288     }
289 
290     public InputStream getInputStream() throws IOException {
291         assertConnected();
292 
293         return this.conn.getInputStream();
294     }
295 
296     public OutputStream getOutputStream() throws IOException {
297         assertConnected();
298 
299         return this.conn.getOutputStream();
300     }
301 
302     public int getResponseCode() throws IOException {
303         assertConnected();
304 
305         return this.conn.getResponseCode();
306     }
307 
308     public String getResponseMessage() throws IOException {
309         assertConnected();
310 
311         return this.conn.getResponseMessage();
312     }
313 
314     public void requestCredDeleg(final boolean requestDelegation) {
315         this.assertNotConnected();
316 
317         this.reqCredDeleg = requestDelegation;
318     }
319 
320     public void setRequestMethod(final String method) {
321         assertNotConnected();
322 
323         this.requestMethod = method;
324     }
325 
326     public SpnegoAuthScheme getAuthScheme(final String header) {
327 
328         if (null == header || header.isEmpty()) {
329             System.out.println("authorization header was missing/null");
330             return null;
331 
332         } else if (header.startsWith(Constants.NEGOTIATE_HEADER)) {
333             final String token = header.substring(Constants.NEGOTIATE_HEADER.length() + 1);
```

```
334             return new SpnegoAuthScheme(Constants.NEGOTIATE_HEADER, token);
335
336         } else if (header.startsWith(Constants.BASIC_HEADER)) {
337             final String token = header.substring(Constants.BASIC_HEADER.length() + 1);
338             return new SpnegoAuthScheme(Constants.BASIC_HEADER, token);
339
340         } else {
341             throw new UnsupportedOperationException("Negotiate or Basic Only:" + header);
342         }
343     }
344
345     private GSSName getServerName(final URL url) throws GSSException {
346 //         return MANAGER.createName("HTTP@" + url.getHost(),
347 //             GSSName.NT_HOSTBASED_SERVICE, getOid());
348             return MANAGER.createName("HTTP/" + url.getHost(), GSSName.NT_USER_NAME);
349     }
350
351
352
353 //             final GSSContext clientContext = dmanager.createContext(gssServerName.➥
    canonicalize(spnegoMechOid), spnegoMechOid, gcred, GSSContext.DEFAULT_LIFETIME);
354
355
356     private Oid getOid() {
357         Oid oid = null;
358         try {
359             oid = new Oid("1.3.6.1.5.5.2");
360             //oid = new Oid("1.2.840.113554.1.2.2");
361         } catch (GSSException gsse) {
362             System.out.println("Unable to create OID 1.3.6.1.5.5.2 !" + gsse);
363             gsse.printStackTrace();
364             //LOGGER.log(Level.SEVERE, "Unable to create OID 1.2.840.113554.1.2.2 !", gsse);
365         }
366         return oid;
367     }
368
369     public GSSContext getGSSContext(final GSSCredential creds, final URL url)
370         throws GSSException {
371
372         return MANAGER.createContext(getServerName(url).canonicalize(getOid())
373                 , getOid()
374                 , creds
375                 , GSSContext.DEFAULT_LIFETIME);
376     }
377
378     public GSSContext getGSSContextWithDefaultOID(final GSSCredential creds, final URL url)
379         throws GSSException {
380
381         return MANAGER.createContext(getServerName(url).canonicalize(null)
382                 , null
383                 , creds
384                 , GSSContext.DEFAULT_LIFETIME);
385     }
386
387 }
```

```
1  // 4-8. programlista:
2
3  package hu.alerant.spnego;
4
5  import java.io.ByteArrayOutputStream;
6  import java.io.IOException;
7  import java.net.MalformedURLException;
8  import java.net.URL;
9  import java.security.PrivilegedActionException;
```

```
10
11  import javax.security.auth.login.LoginException;
12  import javax.xml.soap.MessageFactory;
13  import javax.xml.soap.MimeHeaders;
14  import javax.xml.soap.SOAPConnection;
15  import javax.xml.soap.SOAPConstants;
16  import javax.xml.soap.SOAPException;
17  import javax.xml.soap.SOAPMessage;
18
19  import org.ietf.jgss.GSSCredential;
20  import org.ietf.jgss.GSSException;
21
22  public class SpnegoSOAPConnection extends SOAPConnection {
23
24      private final transient SpnegoHttpURLConnection conn;
25
26      public SpnegoSOAPConnection() throws Exception {
27          super();
28          this.conn = new SpnegoHttpURLConnection();
29      }
30
31      @Override
32      public final SOAPMessage call(final SOAPMessage request, final Object endpoint)
33          throws SOAPException {
34
35          SOAPMessage message = null;
36          final ByteArrayOutputStream bos = new ByteArrayOutputStream();
37
38          try {
39              final MimeHeaders headers = request.getMimeHeaders();
40              final String[] contentType = headers.getHeader("Content-Type");
41              final String[] soapAction = headers.getHeader("SOAPAction");
42
43              // build the Content-Type HTTP header parameter if not defined
44              if (null == contentType) {
45                  final StringBuilder header = new StringBuilder();
46
47                  if (null == soapAction) {
48                      header.append("application/soap+xml; charset=UTF-8;");
49                  } else {
50                      header.append("text/xml; charset=UTF-8;");
51                  }
52
53                  // not defined as a MIME header but we need it as an HTTP header parameter
54                  this.conn.addRequestProperty("Content-Type", header.toString());
55              } else {
56                  if (contentType.length > 1) {
57                      throw new IllegalArgumentException("Content-Type defined more than once.");
58                  }
59
60                  // user specified as a MIME header so add it as an HTTP header parameter
61                  this.conn.addRequestProperty("Content-Type", contentType[0]);
62              }
63
64              // specify SOAPAction as an HTTP header parameter
65              if (null != soapAction) {
66                  if (soapAction.length > 1) {
67                      throw new IllegalArgumentException("SOAPAction defined more than once.");
68                  }
69                  this.conn.addRequestProperty("SOAPAction", soapAction[0]);
70              }
71
72              request.writeTo(bos);
73
74              this.conn.connect(new URL(endpoint.toString()), bos);
```

```
75
76              final MessageFactory factory = MessageFactory.newInstance(
77                      SOAPConstants.SOAP_1_2_PROTOCOL);
78
79              try {
80                  message = factory.createMessage(null, this.conn.getInputStream());
81              } catch (IOException e) {
82                  message = factory.createMessage(null, this.conn.getErrorStream());
83              }
84
85          } catch (MalformedURLException e) {
86              throw new SOAPException(e);
87          } catch (IOException e) {
88              throw new SOAPException(e);
89          } catch (GSSException e) {
90              throw new SOAPException(e);
91          } catch (PrivilegedActionException e) {
92              throw new SOAPException(e);
93          } finally {
94              try {
95                  bos.close();
96              } catch (IOException ioe) {
97                  assert true;
98              }
99              this.close();
100         }
101
102         return message;
103     }
104
105     @Override
106     public final void close() {
107         if (null != this.conn) {
108             this.conn.disconnect();
109         }
110     }
111 }
```

Amennyiben a Kerberos alapú hitelesítés nem sikerült, úgy a *web.xml* deklarációja szerint FORM alapú autentikációra teszünk kísérletet a *login.jsp* lap használatával:

```
1  // 4-9. programlista: login.jsp
2
3  <!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
4  <html>
5  <META HTTP-EQUIV = "Pragma" CONTENT="no-cache">
6  <title> Login Page </title>
7  <body>
8  <h2>Form Login</h2>
9  <form method=POST action="j_security_check">
10 <p>
11 <font size="2"> <strong> Enter user ID and password: </strong></font>
12 <BR>
13 <strong> User ID</strong> <input type="text" size="20" name="j_username">
14 <strong> Password </strong>  <input type="password" size="20" name="j_password">
15 <BR>
16 <BR>
17 <font size="2">  <strong> And then click this button: </strong></font>
18 <input type="submit" name="login" value="Login">
19 </p>
20
21 </form>
22 </body>
23 </html>
```

```
1  // 4-10. programlista: error.jsp
2
3  <!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.0 Transitional//EN">
4  <html>
5  <head><title>A Form login authentication failure occurred</head></title>
6  <body>
7  <H1><B>A Form login authentication failure occurred</H1></B>
8  <P>Authentication might fail for one of many reasons. Some possibilities include:
9  <OL>
10 <LI>The user ID or password might have been entered incorrectly; either misspelled or the
11 wrong case was used.
12 <LI>The user ID or password does not exist, has expired, or has been disabled.
13 </OL>
14 </P>
15
16 <%new Exception().printStackTrace();%>Exception
17
18 </body>
19 </html>
```

Összefoglalás és Irodalomjegyzék

Ebben a cikkben az AD-t tartalmazó Windows szerver környezetet, Weblogic szervert és a Kerberost használtuk együtt. A cél az volt, hogy az elterjedten használt Windows Integrated hitelesítést (amikor a Windows desktoppal bejelentkeztünk egy Windows domain-be) használhassuk. Erre sokáig az NTLM-et használtuk, de manapság inkább a Kerberos használata javasolt. Remélhetőleg ez az összefoglaló segít bennünket elindulni ezen az úton. Segítségül szeretnénk megosztani néhány további érdekes olvasnivalót.

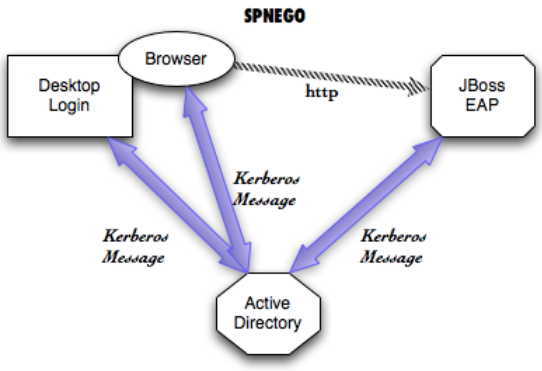
- Configuring JBoss for Windows Integrated Authentication: `http://spnego.sourceforge.net/spnego_jboss.html`
- SPNEGO SSO administrators guide for JBoss
- Integrated Windows Authentication in Java: `http://spnego.sourceforge.net/`
- SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows: `http://tools.ietf.org/html/rfc4559`
- HTTP-Based Cross-Platform Authentication by Using the Negotiate Protocol: `http://msdn.microsoft.com/en-us/library/ms995330.aspx`
- Apache Kerberos támogatás: `http://modauthkerb.sourceforge.net/`
- Java Generic Security Services (Java GSS) and Kerberos: `http://docs.oracle.com/javase/6/docs/technotes/guides/security/jgss/jgss-features.html`
- EJB3 Authentication With SPNEGO: `https://community.jboss.org/wiki/EJB3AuthenticationWithSPNEGO`

***

Befejezésül szeretnénk megköszönni Darmai Gábor (Alerant Informatikai Zrt. - technológiai igazgató) támogatását, aki nélkül ez a cikk nem készülhetett volna el.

5. A SPNEGO SSO beállítása JBoss környezetben

A előző cikkben az Oracle Weblogic környezetre tekintettük át a Windows Integrated SPNEGO SSO használatát, ott sok mindent megtanultunk. Most a népszerű JBoss alkalmazásszerverre tekintjük át ugyanezt, hiszen ez egy komoly és rendkívül megbízható alternatívája a fizetős JEE környezeteknek.



5.1. ábra. JBoss, AD, Kerberos és SPNEGO

Az 5.1. ábra a már ismerős Kerberos kommunikációs sémát mutatja, ezért induljunk ki ismét ebből az alapműködésből és nézzük meg röviden, hogy a JBoss szerveren milyen beállítási lépések szükségesek. Tekintsünk egy ilyen url-t: `http://jbserver.ceg.hu`. Ekkor gépünk a *ceg.hu* domainban van, a gép neve pedig *jbserver* (az jboss szerver), amire a kerberos beállításokat el fogjuk végezni.

`http://sourceforge.net/projects/spnego/files/`

Az előfeltételek áttekintése

Az előző cikkből sok mindent megtanultunk, de ennek ellenére egy gyors ellenőrző lista hasznos lehet.

A kliens gépek ellenőrzése

A böngészőt futtató munkaállomásnak egy megfelelő Windows domain-ban kell lennie, különben az automatikus SPNEGO hitelesítés nem tud működni. Erről az adott Windows gépen könnyen meggyőződhetünk, ha megnyitjuk a System Properties ablakot (Start→Settings→Control Panel), ahol 2 fontos információt találunk:

1. A számítógép teljes neve (FQDN, példa: *gepem.ceg.hu*) és
2. A domain neve

Lényeges az is, hogy a gépet éppen használó user a domain-ba legyen bejelentkezve.

Windows környezet és domain account

Erről a Weblogicról szóló cikk *Windows Server oldali konfiguráció* részében részletesen írtunk, az ott olvashatóak természetesen itt is érvényesek. Esetünkben a Windows domain adminisztrátorral közösen elkészített SPN ilyen alakú lesz:

```
http/jbserver.ceg.hu@CEG.HU
```

```
setspn.exe -a HTTP/jbserver.ceg.hu@CEG.HU ➥
    jbossuser
```

```
setspn.exe -l jbossuser
```

```
ktpass -princ HTTP/jbserver.ceg.hu@CEG.HU -➥
    pass * -mapuser CEG\jbossuser -out
c:\jbossuser.http.keytab
```

```
ktab -k c:\jbossuser.http.keytab -a ➥
    jbossuser@CEG.HU
```

A Firefox beállítása

Az előző írásban az IE beállítása szerepelt, itt most emiatt csak a Firefox szükséges beállítását adjuk meg. Az ismert *about:config* URL segítségével a következő 2 beállítás szükséges:

- *network.negotiate-auth.trusted-uris* → *http://,https://*
- *network.negotiate-auth.delegation-uris* → *http://,https://*

A spnego.jar telepítése

A Weblogic 11g gyárilag tartalmazott minden szükséges komponenst a SPNEGO Kerberos kialakításához, azonban JBOSS esetén töltsük le a *spnego.jar* file-t innen: `http://sourceforge.net/projects/spnego/files/` (itt a cikk írásakor a *spnego-r7.jar* és a *spnego-r5.jar* volt elérhető). Mi az r7 változatot használtuk, amit ide kell másolni, mert mi a *default* domaint használjuk: *JBOSS_HOME/server/default/lib*.

A krb5.conf file beállítása

Állítsuk le a JBOSS szervert! A kerberos *krb5.conf* beállításának lépései megegyeznek a Weblogic-nál írtakkal, azonban azt most a *JBOSS_HOME/bin* könyvtárba kell elhelyeznünk. Fontos, hogy a JRE képes legyen ezt a file-t megtalálni.

A login-config.xml file beállítása

A *JBOSS_HOME/server/default/conf* könyvtárban lévő *login-config.xml* file-ba a következő sorokat kell elhelyezni:

```
...
<application-policy name="spnego-client">
      <authentication>
        <login-module code="com.sun.security.➥
            auth.module.Krb5LoginModule"
          flag="required" />
      </authentication>
    </application-policy>

    <application-policy name="spnego-server">
      <authentication>
        <login-module code="com.sun.security.➥
            auth.module.Krb5LoginModule"
          flag="required">
          <module-option name="storeKey">true<➥
              /module-option>
        </login-module>
      </authentication>
    </application-policy>
...
```

Tesztelés

A tesztelést ezzek a kis jsp lappal végezhetjük el, ami kiírja a SPNEGO token alapján elérhető hitelesített felhasználó nevét.

```
<html>
<head>
    <title>Hello SPNEGO Example</title>
</head>
<body>
Hello <%= request.getRemoteUser() %> !
</body>
</html>
```

A használt *web.xml* file:

```
// web.xml

<filter>
    <filter-name>SpnegoHttpFilter</filter-name>
    <filter-class>net.sourceforge.spnego.SpnegoHttpFilter</filter-class>
    <init-param>
        <param-name>spnego.allow.basic</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.allow.localhost</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
```

```
        <param-name>spnego.allow.unsecure.basic</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.login.client.module</param-name>
        <param-value>spnego-client</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.krb5.conf</param-name>
        <param-value>krb5.conf</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.login.conf</param-name>
        <param-value>login.conf</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.preauth.username</param-name>
        <param-value>Zeus</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.preauth.password</param-name>
        <param-value>Z3usP@55</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.login.server.module</param-name>
        <param-value>spnego-server</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.prompt.ntlm</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>spnego.logger.level</param-name>
        <param-value>1</param-value>
    </init-param>
</filter>

<filter-mapping>
    <filter-name>SpnegoHttpFilter</filter-name>
    <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

Amennyiben valaki a gyakorlatban is végig szeretné próbálni a leírtakat, tanulmányozza ezt a webhelyet: `http://spnego.sourceforge.net/spnego_jboss.html`

6. A JBOSS – Bonita páros produktív használata

Ez a cikk röviden ismerteti a Bonita BPM motor JBOSS Java szerver környezetre való konfigurációjának lépéseit. Ez nagyrészt az adatbázis perzisztencia réteg beállítását jelenti, ezért itt most ezt mutatjuk be Oracle adatbázis-kezelőt használva.

Konfigurálás Oracle adatbázishoz

A Bonita környezettel előre összecsomagolt letölthető JBOSS verziót (jelen esetben 5.1.0 GA, letölthető innen: `http://www.bonitasoft.com/products/download/jboss-5.1.0-9`) egy *server.company.org* (továbbiakban: *server*) nevű gép */opt/BOS-5.5.2-JBoss-5.1.0.GA* könyvtárába (továbbiakban erre *jboss_home* néven fogunk hivatkozni) csomagoltuk ki és a JBOSS *default* domain-ját használjuk. Feltételezzük, hogy a Java 1.6 már működik az operációs rendszeren. A JBOSS működőképességét próbáljuk ki úgy, hogy elindítjuk (A *-b 0.0.0.0* azt jelenti, hogy más számítógépekről is elérhető lesz az alkalmazásszerver, ugyanis nem ez az alapértelmezés):

```
./<jboss_home>/run.sh -b 0.0.0.0 &
```

A JBOSS szerver leállítása (a *-s* után a JNDI Naming Service-re hivatkozó URL van):

```
./<jboss_home>/shutdown.sh -s jnp://➥
    localhost:1099
```

A következő feladat a JBOSS átkapcsolása Oracle adatbázisra, aminek lépéseit mutatjuk meg az alábbiakban.

A beépített *HSQDB* visszamozgatása

A következő 4 művelet kitörli a hsqdb-vel kapcsolatos file-okat a megfelelő JBOSS könyvtárakból:

```
remove hsqldb.jar from <jboss_home>/common/lib➥
    /
remove hsqldb-plugin.jar from <jboss_home>/➥
    common/lib/
remove hsqldb-ds.xml from <jboss_home>/server➥
    /<domain>/deploy/
remove hsqldb-persistence-service.xml from <➥
    jboss_home>/server/<domain>/deploy/➥
    messaging/
```

Az Oracle használata

Több Oracle thin driver létezik, de ebben a leírásban az *ojdbc6.jar* használata mellett döntöttünk, amit a `http://www.oracle.com/technetwork/indexes/downloads/index.html` helyről lehet hivatalosan letölteni. Ezután ezt a 3 lépést kell végrehajtani:

```
(1) copy ojdbc6.jar to <jboss_home>/server/<➥
    domain>/lib/
(2) create JBosss datasource file (oracle-ds.➥
    xml)
(3) copy oracle-ds.xml to <jboss_home>/server➥
    /<domain>/deploy/
```

Ha szükséges az éppen használt Oracle verzió gyors lekérdezése, akkor ezt az ismert

```
select * from v$version
```

paranccsal tehetjük meg. A fent hivatkozott *oracle-ds.xml* az ismert datasource létrehozási módszer szerinti tartalommal rendelkező XML, amit a 6-1. Programlista mutat. A *oraserver.company.org* az adatbázis szerver neve, a *test* pedig az adatbázis. Amikor az XML-t telepítjük az JBOSS szerverre, esetünkben létrejön egy *DefaultDS* nevű JEE Datasource. Ez az a JNDI név, amit a programozó is használ, illetve a konfigurációkban is kulcszerepe van, hiszen ezen keresztül tudunk adatbázis connection-okat szerezni.

```
// 6-1. Programlista

<?xml version="1.0" encoding="UTF-8"?>

<datasources>
   <local-tx-datasource>
      <!-- The jndi name of the DataSource, it is prefixed with java:/ -->
      <jndi-name>DefaultDS</jndi-name>

      <connection-url>jdbc:oracle:thin:@oraserver.company.org:1521:test</connection-url>

      <!-- The driver class -->
      <driver-class>oracle.jdbc.driver.OracleDriver</driver-class>

      <!-- The login and password -->
      <user-name>bonita_history</user-name>
      <password>BoHis_1315</password>
     <exception-sorter-class-name>org.jboss.resource.adapter.jdbc.vendor.OracleExceptionSorter</➥
         exception-sorter-class-name>

         <!-- this will be run before a managed connection is removed from the pool for use by a➥
              client -->
      <check-valid-connection-sql>select 1 from dual</check-valid-connection-sql>

      <!-- The minimum connections in a pool/sub-pool. Pools are lazily constructed on first use➥
           -->
      <min-pool-size>5</min-pool-size>

      <!-- The maximum connections in a pool/sub-pool -->
      <max-pool-size>20</max-pool-size>

      <!-- The time before an unused connection is destroyed -->
      <!-- NOTE: This is the check period. With <idle-timeout-minutes> you can indicate the ➥
          maximum time a connection
           may be idle before being closed and returned to the pool. If not specified it's 15 ➥
               minutes.
          -->
      <idle-timeout-minutes>5</idle-timeout-minutes>

      <!-- sql to call when connection is created -->
      <new-connection-sql>select 1 from dual</new-connection-sql>

      <!-- sql to call on an existing pooled connection when it is obtained from pool -->
      <check-valid-connection-sql>select 1 from dual</check-valid-connection-sql>

      <!-- example of how to specify a class that determines a connection is valid before it is ➥
          handed out from the pool
         <valid-connection-checker-class-name>org.jboss.resource.adapter.jdbc.vendor.➥
             DummyValidConnectionChecker</valid-connection-checker-class-name>
      -->

      <!-- Whether to check all statements are closed when the connection is returned to the ➥
          pool,
           this is a debugging feature that should be turned off in production -->
      <track-statements>false</track-statements>

      <!-- Use the getConnection(user, pw) for logins
        <application-managed-security/>
      -->

      <!-- Use the security domain defined in conf/login-config.xml -->
      <security-domain>OracleDbRealm</security-domain>

      <!-- Use the security domain defined in conf/login-config.xml or the
```

```
            getConnection(user, pw) for logins. The security domain takes precedence.
           -->
      <security-domain-and-application>OracleDbRealm</security-domain-and-application>

      <!-- This element specifies the number of prepared statements per connection in an LRU ➥
           cache, which is keyed
            by the SQL query. Setting this to zero disables the cache.
           -->
      <prepared-statement-cache-size>32</prepared-statement-cache-size>
      <metadata>
         <type-mapping>Oracle9i</type-mapping>
      </metadata>

   </local-tx-datasource>

</datasources>
```

A login-config.xml beállítása

A JBOSS login beállítás a

```
<jboss_home>/server/<domain>/conf/login-config➥
    .xml
```

file-ban található. Az itt lévő application policy neve alapértelmezésben a *HsqlDbRealm*, amit át kell állítani *OracleDbRealm*-re, a 6-2. Programlistán mutatott módon.

A perzisztencia beállítása

A JBOSS több mintafájlt tartalmaz, mi innen vegyük el a következőt:

```
<jboss_home>\docs\examples\jms\oracle-➥
    persistence-service.xml
```

és másoljuk ide:

```
<jboss_home>/server/<domain>/deploy/messaging/
```

```
// 6-2. Programlista

<application-policy name="OracleDbRealm">
        <authentication>
                <login-module code="org.jboss.resource.security.ConfiguredIdentityLoginModule" ➥
                    flag="required">
                        <module-option name="principal">test</module-option>
                        <module-option name="userName">bonita_history</module-option>
                        <module-option name="password">BoHis_1315</module-option>
                        <module-option name="managedConnectionFactoryName">jboss.jca:service=➥
                            LocalTxCM,name=DefaultDS</module-option>
                </login-module>
        </authentication>
</application-policy>
```

Indítsuk el a JBOSS-t és nézzük meg a legenerált új táblákat!

A Bonita BPM Oracle beállításai

Eddig áttekintettük azt a JBOSS Oracle perzisztenciájának kialakítását, most ugyanebbe az adatbázisba végezzük el a Bonitával összefüggő konfigurációt. A Bonita ajánlása az, hogy 2 sémát használjunk. Az egyik a journal, a másik history célra szolgál. Ennek megfelelően a sémáink a már ismert adatbázisban a következőek:

```
host: oraserver.company.org (port: 1521)
sid: test
user 1: bonita_journal/BonJour_1316
user 2: bonita_history/BoHis_1315
```

Ezután állítsuk be a *bonita-journal.properties* és *bonita-history.properties* file-okat az alábbiak szerint (látható, hogy a BPM perzisztencia kezeléshez a Bonita a *Hibernate* könyvtárat használja):

```
<bonita_home>/bonita/server/default/conf/bonita-journal.properties should be like:
        #hibernate.hbm2ddl.auto                    update
         hibernate.dialect                         org.hibernate.dialect.Oracle10gDialect
         hibernate.connection.driver_class         oracle.jdbc.OracleDriver
         hibernate.connection.url                  jdbc:oracle:thin:@oraserver.company.org:1521:➥
             test
         hibernate.connection.username             bonita_journal
         hibernate.connection.password             BonJour_1316
         hibernate.connection.shutdown             true
         hibernate.cache.use_second_level_cache    false
         hibernate.cache.use_query_cache           false
         hibernate.show_sql                        false
         hibernate.format_sql                      false
         hibernate.use_sql_comments                false
         bonita.search.use                         true
         hibernate.search.default.indexBase        ${BONITA_HOME}/server/default/work/indexes/➥
             journal

<bonita_home>/bonita/server/default/conf/bonita-history.properties should be like:
        #hibernate.hbm2ddl.auto                    update
         hibernate.dialect                         org.hibernate.dialect.Oracle10gDialect
         hibernate.connection.driver_class         oracle.jdbc.OracleDriver
         hibernate.connection.url                  jdbc:oracle:thin:@oraserver.company.org:1521:➥
             test
         hibernate.connection.username             bonita_history
         hibernate.connection.password             BoHis_1315
         hibernate.connection.shutdown             true
         hibernate.cache.use_second_level_cache    false
         hibernate.cache.use_query_cache           false
         hibernate.show_sql                        false
         hibernate.format_sql                      false
         hibernate.use_sql_comments                false
         bonita.search.use                         true
         hibernate.search.default.indexBase        ${BONITA_HOME}/server/default/work/indexes/➥
             history
```

Ezután a Bonita webhelyéről töltsük le a *BOS-5.7.2-deploy.zip* file-t (ez a legfrissebb verzió a cikk írásakor) innen: `http://www.bonitasoft.com/products/BPM_downloads`. Csomagoljuk ki egy alkalmas könyvtárba, aminek a gyökerét *BOS-5.7.2-deploy*-nak nevezzük a továbbiakban. A most kialakított 2 properties file-t másoljuk ide:

```
<BOS-5.7.2-deploy>\conf\bonita\server\default\➥
    conf\
```

Ezután futtassuk az *initDatabase.sh* scriptet, ami itt található:

```
<BOS-5.7.2-deploy>/bonita_execution_engine/➥
    database/
```

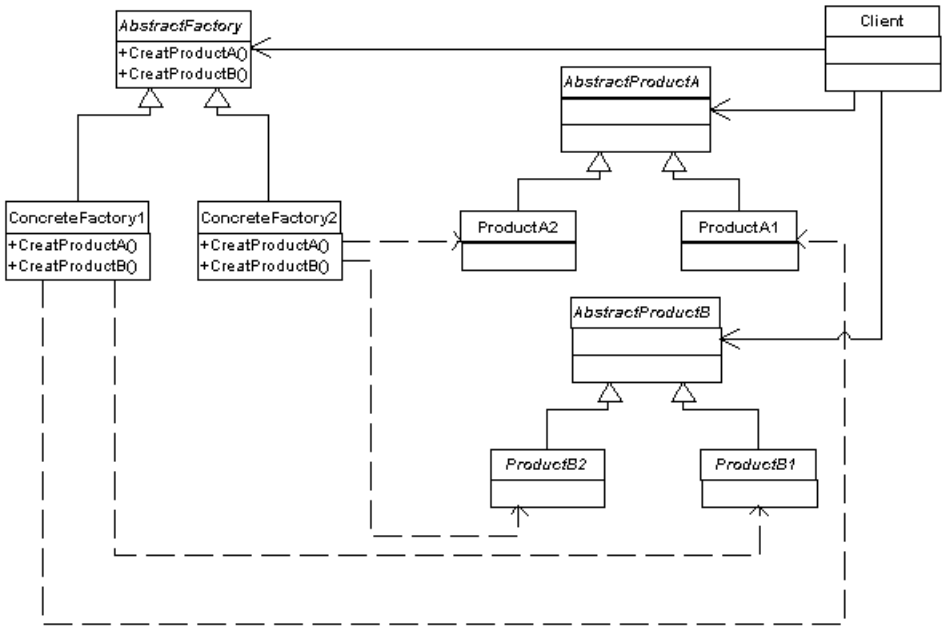
A következőkben a script ezeket megkérdezi mielőtt elkezdi az adatbázisban létrehozni a Bonita releváns objektumokat:

```
Which domain do you want to use (press enter ➥
    without nothing to use default)?
[PRESS ENTER]

Where is your BONITA_HOME folder?
<BOS-5.6.2-deploy>/conf/bonita/

Which hibernate configuration to use to ➥
    generate database (press enter without ➥
    nothing to use default)?
[PRESS ENTER]
```

A futás után a használt Bonita verzióban 69 darab új tábla jött létre a fenti 2 sémában.

7. Az Abstract Factory minta

Képzeljük el, hogy van egy komponenskészletünk, amiben az egyes komponenseket jól meghatározott interface-en keresztül érjük el. Amikor egy alkalmazást készítünk ezekkel a komponensekkel, akkor azok beépítésekor csak az interface-eik által nyújtott szolgáltatásokat használjuk ki. Emiatt készíthetünk több hasonló, de másik komponens készletet is, aminek a használatára csak egy egyszerű átkapcsolással szeretnénk áttérni. Ilyenkor mindenképpen javasoljuk az Abstract Factory tervezési minta szerinti felépítést.

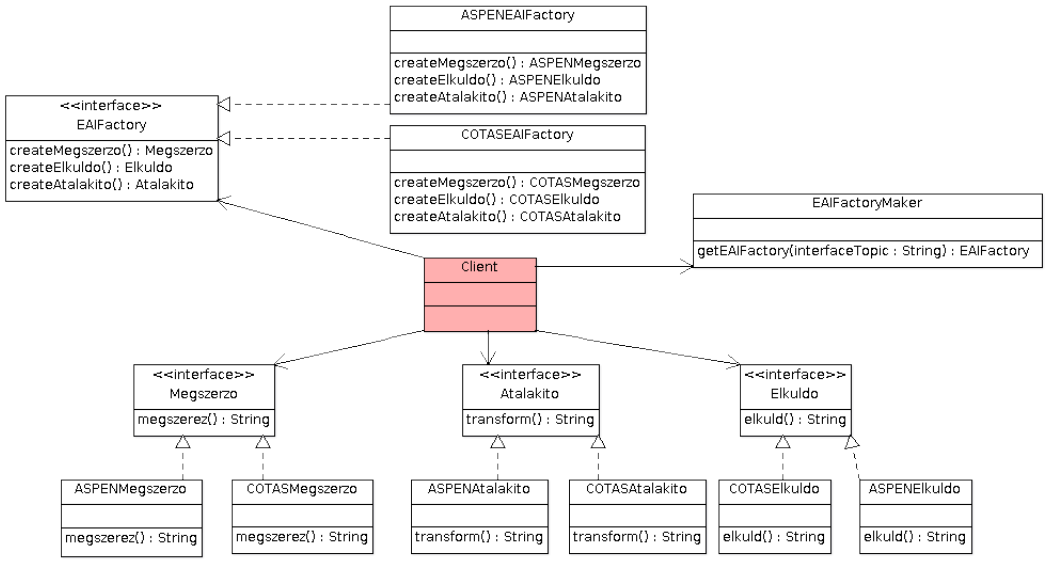


7.1. ábra. Az Abstract Factory minta UML diagramja

Áttekintés

Nézzünk egy példát, amikor tipikusan az Abstract Factory minta használatos! Van egy GUI komponenskészletünk, ami ilyesmi komponensekből áll: Button, RadioButton, CheckBox, List, Text, MultiText, stb. Mindegyik elemet egy számára szabványosított felületen éri el a kliens, azaz az őt használó program (Ő egy másik osztály). Felépítünk egy GUI felületet ezekből az elemekből és minden tökéletesen működik. Később másfajta keretrendszerek is megjelennek, újfajta gombokkal, listákkal és egyéb elemekkel. Az egyszerűség kedvéért tegyük fel, hogy ezek interface is hasonló, illetve némi munkával hasonlóvá tehető. Szeretnénk, ha a programunk használni tudná ezt az új, esetleg webes vagy más futtató környezetben is jól működő keretrendszert. A gond az, hogy a kliens programunkban mindenütt „beégetve” használtuk az először megjelent komponenskészlet vezérlőinek típusneveit? Milyen jó lenne, ha a kliens GUI csak interface-en keresztül függne a konkrét vezérlőktől és emiatt gyorsan át tudnánk kapcsolni

az egyes ablakozó felületek között. Nézzünk csak a 7.1. ábrára, gondolkodjunk el és remélhetőleg felismertük, hogy ebben az Abstract Factory minta tud segíteni. Figyeljük meg mi a kapcsolat az ábra és a mi mostani konkrét feladatunk között! Ami az ábrán az *AbstractProductA*, *AbstractProductB*, az a mi esetünkben a *Button*, *RadioButton*, *CheckBox*, *List*, *Text*, *MultiText*, ... Amikor egy Motif, Java Swing, KDE, GNOME, Windows Forms típusú megvalósításokat használunk, akkor például *MotifButton*, *MotifRadioButton*, *MotifCheckBox*, *MotifList*, *MotifText*, *MotifMultiText* vagy *KDEButton*, *KDERadioButton*, *KDECheckBox*, *KDEList*, *KDEText*, *KDEMultiText* a konkrét product. Minden komponens család egy *AbstractFactory* felületű factory objektum segítségével gyártható le úgy, hogy először a család konkrét factory objektumát szerezzük meg. A minta előnye, hogy a kliens nem is tudja, hogy milyen konkrét keretrendszer dolgozik a háttérben.



7.2. ábra. Az EAI komponensek

Példa - EAI komponensek

Az EAI[3] megoldások feladata, hogy az egyes alkalmazások között kiépített interface-ekkel biztosítsuk a közöttük való együttműködés lehetőségét, amivel azután munkafolyamatok is kiépíthetőek. Tipikus feladat, hogy az egyes alkalmazásoktól megszerezzük az adatokat, átalakítjuk egy szabványos (például XML) formátumra, majd elküldjük az üzenetkezelő rendszer részére. Ilyenkor a kliens alkalmazás egy *input adapter*, ami az alkalmazást kifelé irányban összeköti a világgal. Egy ilyen feladatban persze nem grafikus vezérlők vannak, hanem adat megszerzők, XML alakra való átalakítók és elküldő komponensek. A problémakör modellezését a 7.2. UML ábra mutatja. Képzeljük el, hogy van 3 termékünk: Megszerző, Átalakító és Elküldő. Ezek több dol-

[3]EAI=Enterprise Application Integration

got is csinálhatnak, de most a példánkban mindegyiknek csak 1 darab metódusa lesz. Egy valódi feladatban az a tipikus, hogy több van, hiszen megszerezni is több módon lehet, annak a körülményeit (forrás kódlap, stb.) be kell állítani. A példánkban most nem absztrakt osztályokat, hanem interface-t használunk ezek leírására, nézzük meg mindhármat:

```
package cs.test.dp.abstractfactory;

public interface Megszerzo
{
    public String megszerez();
}
```

```
package cs.test.dp.abstractfactory;

public interface Atalakito
{
    public String transform();
}
```

```
package cs.test.dp.abstractfactory;

public interface Elkuldo
{
    public String elkuld();
}
```

A kérdés az, hogy ki gyártja le nekünk azokat az objektumokat, amik ilyen interface-szel rendelkeznek. Ennek a tervezési mintának az a jellegzetessége, hogy első körben ezt is elvont módon tesszük, azaz az *EAIFactory* is csak egy interface, az olyan factory objektumok felülete, amik képesek nekünk *Megszerzo*, *Atalakito* vagy *Elkuldo* példányokat gyártani.

```
package cs.test.dp.abstractfactory;

public interface EAIFactory
{
    public Megszerzo createMegszerzo();
    public Elkuldo createElkuldo();
    public Atalakito createAtalakito();
}
```

A termékeinknek egyelőre 2 családját csináljuk meg, az egyes családok nevei: *ASPEN* és *COTAS*. A későbbiekben mindkét család konkrét termékeit implementálni fogjuk, de itt csak az egyes *EAIFactory* felületeket mutatjuk egyelőre:

```
package cs.test.dp.abstractfactory;

public class ASPENEAIFactory implements ➥
    EAIFactory
{

    public Megszerzo createMegszerzo()
    {
        return new ASPENMegszerzo();
    }

    public Elkuldo createElkuldo()
    {
        return new ASPENElkuldo();
    }

    public Atalakito createAtalakito()
    {
        return new ASPENAtalakito();
    }
}
```

```
package cs.test.dp.abstractfactory;

public class COTASEAIFactory implements ➥
    EAIFactory
{

    public Megszerzo createMegszerzo()
    {
        return new COTASMegszerzo();
    }

    public Elkuldo createElkuldo()
    {
        return new COTASElkuldo();
    }

    public Atalakito createAtalakito()
    {
        return new COTASAtalakito();
    }
}
```

Van tehát 2 termékcsaládunk, de a legyártásukhoz kell egy-egy gyár. Amikor gyárat alapítunk, akkor az *EAIFactoryMaker* classt hívhatjuk segítségül. Például egy *EAIFactory.getEAIFactory(„ASPEN”)* hívással egy ASPEN termékcsaládot gyártó gyárhoz jutunk, aminek persze olyan a felülete, mint minden gyárnak (*EAIFactory*):

```
package cs.test.dp.abstractfactory;

public class EAIFactoryMaker
{
    public static EAIFactory getEAIFactory(➥
        String interfaceTopic)
    {
        if (interfaceTopic.equals("COTAS"))
        {
```

```
            return new COTASEAIFactory();
        }
        else if (interfaceTopic.equals("ASPEN"➥
            ))
        {
            return new ASPENEAIFactory();
        }
        return null;
    }
}
```

Egy gyár dolgozik, előre megtervezett termékek készülnek, így a mintánk implementálása során ezek terve sem maradhat kidolgozatlan. A következő 6 class a 3 ASPEN és 3 COTAS termék gyártási terve. Lehetnének sokkal összetettebbek is, működhetnének bonyolultabban, de talán akkor sem mutatnák be jobban ennek a mintának a lényegét:

```
package cs.test.dp.abstractfactory;

public class ASPENMegszerzo implements ➥
    Megszerzo
{
    public String megszerez()
    {
        return "ASPEN_getter";
    }
}
```

```
package cs.test.dp.abstractfactory;

public class ASPENAtalakito implements ➥
    Atalakito
{

    public String transform()
    {
        return "ASPEN_TR";
    }

}
```

```
package cs.test.dp.abstractfactory;

public class ASPENElkuldo implements Elkuldo
{

    public String elkuld()
    {
        return "ASPEN_sender";
    }
}
```

```
package cs.test.dp.abstractfactory;

public class COTASMegszerzo implements ➥
    Megszerzo
{
    public String megszerez()
    {
        return "COTAS_getter";
    }
}
```

```
package cs.test.dp.abstractfactory;

public class COTASAtalakito implements ➥
    Atalakito
{

    public String transform()
    {
        return "COTAS_TR";
    }

}
```

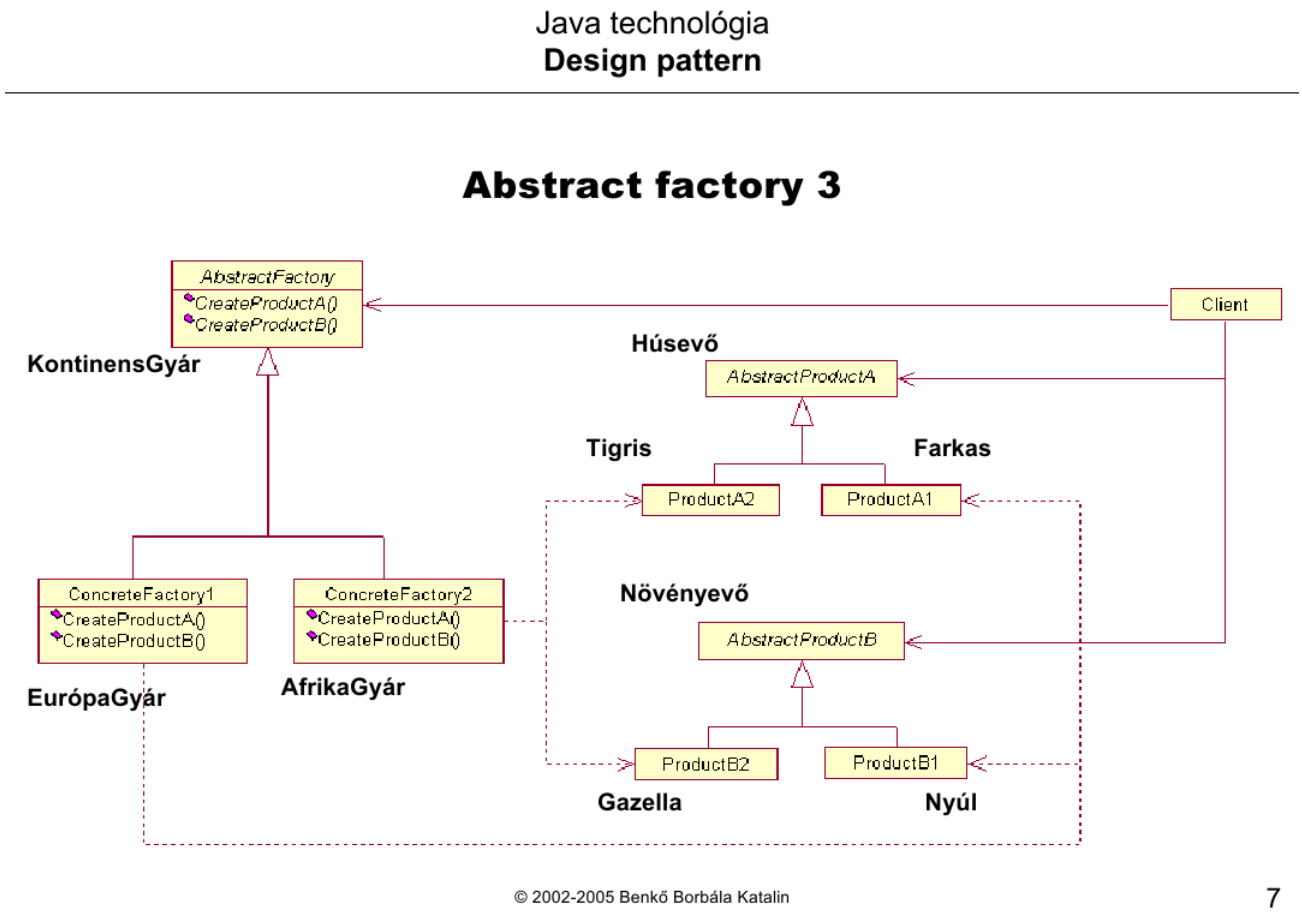
```
package cs.test.dp.abstractfactory;

public class COTASElkuldo implements Elkuldo
{

    public String elkuld()
    {
        return "COTAS_sender";
    }
}
```

Elkészült minden. Rendelkezünk 2 gyárral és 6 termékkel. Itt van az a pont, amikor őket egy alkalmazásban bevethetjük. Ezt a helyet hívjuk más szóval kliens osztálynak (vagy egyszerűen kliensnek), ami a lenti példában a *Main*. Most éppen a COTAS család termékeit használjuk, ezért az *EAIFactoryMaker* osztályt egy ilyen gyár létrehozására kérjük meg. A gyár neve: *fact* lett, ami mindegyik termékből most 1-1 darabot gyárt csak, de ezekre sincs korlát, csinálhatna sokat, de esetenként nullát is, ahogy a pillanatnyi igény hozza. A lenti példakliensünk kicsit egyszerűen ugyan, de elkezdi használni a legyártott termékeket, majd mindegyiket megkéri arra, hogy csinálják azt, amihez értenek: *megszerez()*, *elkuld()*, *transform()*.

```
package cs.test.dp.abstractfactory;

public class Main
{

    public static void main(String[] args)
    {
        System.out.println("Teszt_Abstract_➥
            Factory");
```

```
        EAIFactory fact = EAIFactoryMaker.➥
            getEAIFactory("COTAS");
        //EAIFactory fact = EAIFactoryMaker.➥
            getEAIFactory("ASPEN");

        Megszerzo megszerzo = fact.➥
            createMegszerzo();
        Elkuldo elkuldo = fact.➥
            createElkuldo();
        Atalakito atalakito = fact.➥
            createAtalakito();

        System.out.println( megszerzo.➥
            megszerez() );
        System.out.println( elkuldo.elkuld() )➥
            ;
        System.out.println( atalakito.➥
            transform() );
    }
}
```

Befejezésül a BME egyik segédletéből kölcsönzött példát ajánljuk tanulmányozásra, amit a 7.3. ábra mutat. Gondolkodjunk el rajta! A különféle biológiai környezetek szimulálása egy ilyen osztály felépítéssel remekül megoldható.



7.3. ábra. Egy biológiai program UML modellje