Budapesti Műszaki és Gazdaságtudományi Egyetem
Méréstechnika és Információs Rendszerek Tanszék

Konfigurációs leírók modell alapú fejlesztése

Mérési segédlet

Szolgáltatásbiztonságra tervezés labor (BMEVIMIM236)

Készítette: Horváth Ákos, ahorvath@mit.bme.hu

1 Bevezetés

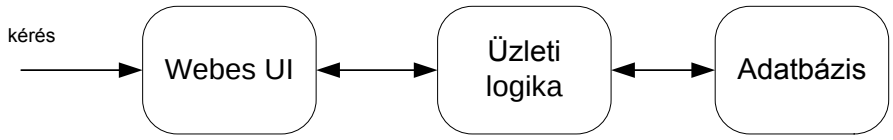
A labor mérései során egy bonyolultabb többrétegű alkalmazást kiszolgáló összetett infrastruktúra szolgáltatásbiztonságát vizsgáljuk különböző technikák segítségével. A rendszer lehetséges felépítési alternatíváit modellezéssel és mérésekkel elemezzük, az így azonosított hibamódok és szűk keresztmetszetek kiküszöbölésére pedig különböző hibatűrést és skálázhatóságot segítő technológiákat próbálunk ki a gyakorlatban. A mérések felépítése a következő:

1. Konfigurációs leírók modell alapú fejlesztése
2. Terheléselosztó fürtök és teljesítménytesztelés
3. Feladatátvételi fürtök
4. Megbízhatóság modellezés
5. Hibadiagnosztika
6. Robosztusság vizsgálata hibainjektálással
7. Szolgáltatásbiztonsági benchmark

Jelen mérés célja, hogy a hallgatók további ismereteket szerezzenek a modell vezérelt fejlesztés lehetséges alkalmazási köréből. Ennek keretében egy magas szintű architektúra leíró nyelvből kiindulva kell konfigurációs leírókat generálniuk, ahol az egyes követelményeket már modell szinten kell validálniuk.

1.1 Háttérismeretek

Webes alkalmazások megvalósítása esetén gyakran használt felépítés a három (vagy finomabb granularitású rendszer esetén az N) rétegű architektúra. Logikai szinten elkülönítjük a főbb funkciókat ellátó rétegeket, úgymint:



Ábra 1 Egy háromrétegű architektúra

A félév során egy „képzeletbeli” 3 rétegű architektúrára épülő rendszer egyes aspektusait fogjuk megvizsgálni, megvalósítani, tesztelni és mérni. Jelen labor keretében a webes UI rétegben megtalálható webszerverekhez befutó kérések elosztásával foglalkozó terheléselosztóhoz (load balancer) kell megfelelő konfigurációs fájlokat generálni. A terheléselosztó feladata, hogy egy előre meghatározott algoritmust követve próbálja meg javítani a rendszer áteresztő képességét olyan módon, hogy a beérkező terhelést egyenletesen szétosztja a rendelkezésre álló erőforrások (a mi esetünkben web kiszolgálók) között.

1.2 Feladat

A mérés során egy olyan eszközt kell elkészíteni, ami képes a *Terheléselosztó fürtök és teljesítmény tesztelé*s mérésben használt terheléselosztó alkalmazás (Apache Tomcat Connector) konfigurációs leíróinak a generálására. Az eszköz *bemenete* egy egyszerű architektúra leíró DSM metamodell, mely a

CIM[1] Core, Network és System sémára épül. Ezt igény szerint ki kell egészíteni, majd a követelményeknek megfelelően validálni kell, végül ebből kell közvetlenül a konfigurációs fájlokat generálni.

A megvalósítás Eclipse alapokon történik, ahol a modellezést EMF-fel kell megvalósítani, a validáláshoz az EMF Validation Framework keretrendszert kell használni, míg a kódgeneráláshoz az OMG MOFMTL (http://www.omg.org/spec/MOFM2T/) szabványt legnagyobb mértékben megvalósító ACCELEO-t kell használni.

2 Technológiák

2.1 Apache Tomcat Connector

Maga az Apache Tomcat egy teljesen Java alapokon megírt webszerver és servlet container. A Connector pedig egy kiegészítés a Tomcathez, amivel terheléselosztó (load balancer) funkciókkal ruházza fel a szervert. Ebben az esetben a terheléselosztó funkcióit betöltő számítógép nem vesz részt a kérések kiszolgálásában, hanem csak a többi *worker* Tomcat menedzselésért felel. A Connectort egy egyszerű szöveges állománnyal lehet konfigurálni. Ennek részletei az alábbi oldalakon érhetőek el:

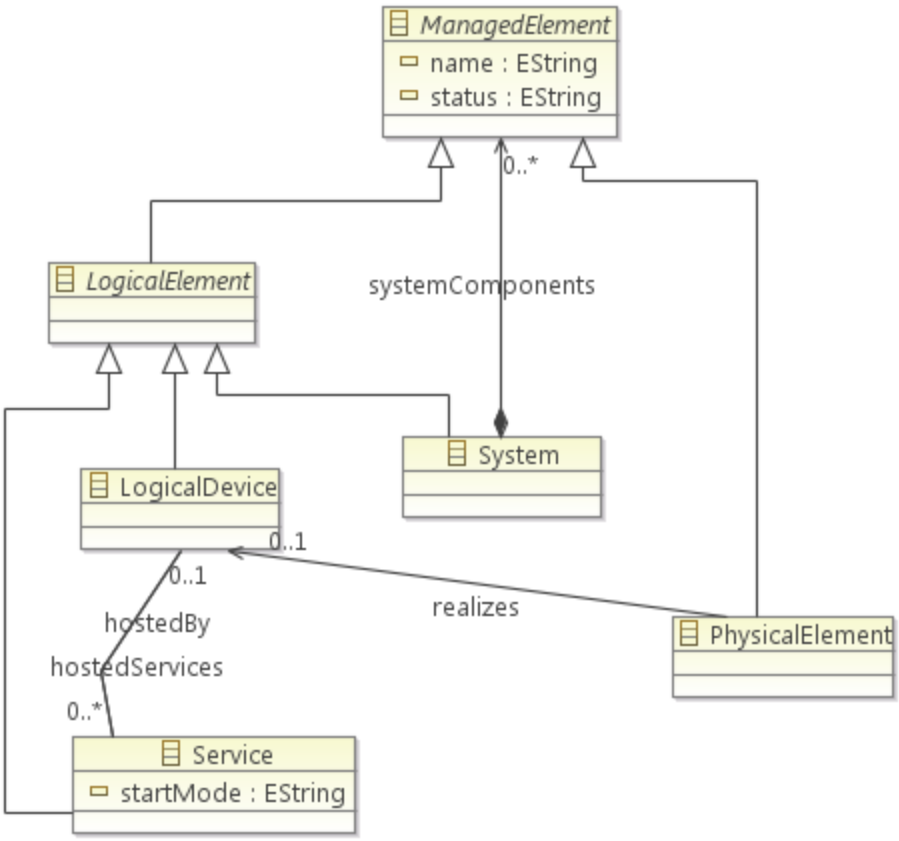
- http://tomcat.apache.org/connectors-doc/generic_howto/workers.html
- http://tomcat.apache.org/connectors-doc/reference/workers.html

Az elkészítendő eszköznek csak az alap konfigurációs módokat kell támogatniuk, azaz a workerek esetén az alábbi attribútumokat: *type, list, host, port, lbfactor, sticky_session, balance_workers.*

2.2 Architektúra leíró DSM

Az architektúra modellezéshez használt nyelv metamodelje az alábbi ábrán (**Hiba! A hivatkozási forrás nem található.**) található. Alapeleme a *ManagedElement,* ami rendelkezik egy név és státusz attribútumokkal. Ebből öröklődik a fizikai szintet reprezentáló *PhysicalElement* és a logikai felépítést összefogó *LogicalElement*. A logikai szint leírására további elemeket definiáltak, úgy mint (i) a *LogicalDevice* aminek a megvalósítását egy konkrét fizikai eszköz fogja végezni, és (ii) a *Service* ami egy a rendszer által nyújtott szolgáltatást reprezentál. Jól látható, hogy a nyelvben megjelent a fizikai és logikai szint elkülönítése, ezzel adva lehetőséget arra, hogy egy már meglévő logikai struktúrát különböző fizikai kiépítésre tudjuk leképezni.

[1] DMTF. Common Information Model, v2.26.0, 2010. URL: http://dmtf.org/standards/cim



2. ábra: DSM Architektúra leíró

Jelenlegi formájában a nyelv **nem képes** az Apache Tomcat Connector specifikus részek modellezésére, ezért kiegészítésre szorul. A kiegészítés során figyeljünk arra, hogy a logikai és fizikai szintek elkülönítését próbáljuk meg megőrizni, és minden esetben csak hozzátegyünk a modellhez, valamint az alap struktúráját adó elemekből ne töröljünk ki semmit.

2.3 EMF Validation Framework

Az Eclipse Modeling Framework lassan a *de facto* (meta)modellező nyelvévé kezd válni az eclipses és tágabb értelemben a javás világnak. Az évek során rengeteg kiegészítő szolgáltatást csatoltak az alap EMF keretrendszerhez. Ezek közül közvetlen a core EMF projekt részévé vált az EMF Validation Framework, ami fejlett nyelvfüggetlen validációs lehetőségeket biztosít EMF példány modellek fölött.

A keretrendszer támogatást nyújt mind *batch* jellegű mind pedig *live* validációs szabályok definiálására. A batch jellegű validáció esetén a felhasználónak kell explicit kiadni a parancsot ennek elvégzésére, míg a live esetben a rendszer úgynevezett change notificationok figyelésével oldja meg a különböző validációs szabályok újraértékelésének az ütemezését.

2.3.1 Megvalósítandó feladat

A labor keretében egy *előre konfigurált* validation pluginbe kell új kényszereket (constraint) felvenni. A pluginnel batch jellegű ellenőrzési módban validálhatunk kényszereket, amiknek a tüzelése a generált

EMF tree Editor Validation gombjára van kikötve. Ahhoz, hogy új kényszereket tudjunk megadni a *org.eclipse.emf.validation.constraintProviders* extension point alatt található *constraints* elem alá kell új *constraint* elemeket létrehozni, és a szükséges paramétereiket kitölteni. Implementációs oldalon pedig az *AbstractModelConstraint* osztályból kell örököltetni a saját kényszereinket.

Az EMF Validation Framework részletes leírása az alábbi címen található: http://help.eclipse.org/helios/index.jsp?nav=/21

2.4 ACCELEO

Az ACCELEO keretrendszer az OMG *Model-to-text* specifikációjának az egyik (lehetséges) megvalósítása. Alapvetően egy sablon (template) alapú kód generátor, amihez fejlett IDE is tartozik olyan funkcionalitásokkal mint: syntax highlighting, content assist, debug (template szinten!), példa alapú sablon készítés stb.

A nyelvezete igen egyszerű, amit az alábbi példa olvashatósága is alátámaszt(hat):

```
[comment encoding = UTF-8 /]
[module generate('/hu.bme.mit.infrastructure/model/infrastructure.ecore')/]

[template public generate(e : System)]

	[comment @main /]
	[file (e.name, false, 'UTF-8')]
	[e.name/]
	Hello ACCELEO World Second attempt
	[for (elem : ManagedElement| e.systemComponents)]
		[if (elem.oclIsKindOf(Service))]
				Service [elem.name/]
		[elseif elem.oclIsKindOf(System)]
				System [elem.name/]
		[elseif elem.oclIsKindOf(PhysicalElement)]
				Physical [elem.name/]
		[elseif elem.oclIsKindOf(LogicalElement)]
				LogicalElement [elem.name/]
		[else]
			Other [elem.name/]
		[/if]
	[/for]
	[/file]

[/template]
```

Mint látható a sablon fejében meghatározható a fájl karakter kódolása. Ezt követően megadhatjuk, hogy mely (meta)modellhez tartozik az aktuális modul. A példánk esetében ez a 2.2-es bekezdésben tárgyalt infrastruktúra metamodell. A sablon nyelv követi az OO alapelveit azaz az egyes modulok mint objektumok és az azokon belül definiált sablonok pedig mint metódusok foghatóak fel. A sablonokhoz lehet elő- és utófeltételeket definiálni, amivel finomabban lehet meghatározni, hogy milyen tulajdonságú elemekre lehessen őket futtatni.

A main annotációval rendelkező modul lesz a kódgenerálás belépő pontja, ez a modul határozza meg a kódgenerálás folyamatát. A rendszer alapvetően az **if-else**, a **let** és **a for** típusú vezérlési szerkezeteket támogatja. Talán elsőre kevésnek tűnhet ez a három elem, de annak fényében, hogy tetszőleges keresést (query) lehet definiálni OCL nyelven, amin utána csak végig akarunk lépkedni, mindjárt megérhető, hogy miért csak ennyi elemet támogatnak.

Az alábbi példa lekérdezés a bemeneti *System* osztály által tartalmazott *systemComponents* listából kigyűjti azon elemeket, amelyeknek a neve megegyezik a *'database'* stringgel.

```
[query public getDatabaseServices(sys : System) : Set(ManagedElement) =
sys.systemComponents->select(name = 'database') /]
```

Az ACCELEO dokumentációja elérhető az alábbi címről:

http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.acceleo.doc/doc/overview.html

3 Ellenőrző kérdések

A mérés elején beugró teszt lesz, a teszt a segédlet és a feladat elvégzéséhez szükséges technológiák ismeretére vonatkozik. Az itt megadott ellenőrző kérdések példaként szolgálnak, hogy milyen kérdésekre lehet számítani, és segítenek abban, hogy felmérjük tudásunkat.

- Milyen validációs módokat támogat az EMF Validation Framework, és ezek miben különböznek?
- Milyen nyelveket támogat az EMF Validation Framework a constraintek felírására?
- ACCELEO template-ek esetén mire lehet használni az elő- és utófeltételeket?
- Mire jó, és milyen funkcionalitást nyújt az automatikusan generálható ACCELEO GUI projekt?
- Írjon fel egy olyan ACCELEO OCL queryt, ami egy osztály összes attribútuma közül visszaadja az összes olyat, amelynek neve hosszabb 10 karakternél és visszatérési értéke boolean. A használt UML class metamodellt is adja meg.

4 Minimum követelmények az elkészítendő programmal kapcsoltban

Annak érdekében, hogy elkerüljük az esetleges félreértéseket, az alábbi pontokat minden a mérés keretében beadott programnak teljesíteni kell ahhoz, hogy azokat elfogadhassuk.

- Exception mentes működés. A program **normál, rendeltetésszerű** működése közben nem dobódhat exception.
- Runtime exception és Error elkapása **tilos** (pl., Null pointer exception).
- Minimális Javadoc a főbb metódusokra.