Általános információk

A diplomaterv szerkezete:

1. Diplomaterv feladatkiírás
2. Címoldal
3. Tartalomjegyzék
4. A diplomatervező nyilatkozata az önálló munkáról és az elektronikus adatok kezeléséről
5. Tartalmi összefoglaló magyarul és angolul
6. Bevezetés: a feladat értelmezése, a tervezés célja, a feladat indokoltsága, a diplomaterv felépítésének rövid összefoglalása
7. A feladatkiírás pontosítása és részletes elemzése
8. Előzmények (irodalomkutatás, hasonló alkotások), az ezekből levonható következtetések
9. A tervezés részletes leírása, a döntési lehetőségek értékelése és a választott megoldások indoklása
10. A megtervezett műszaki alkotás értékelése, kritikai elemzése, továbbfejlesztési lehetőségek
11. Esetleges köszönetnyilvánítások
12. Részletesés pontos irodalomjegyzék
13. Függelék(ek)

Felhasználható a következő oldaltól kezdődő Diplomaterv sablon dokumentum tartalma. Ügyeljen a konzulens nevét és a beadás évét jelölő szövegdobozokra, mert azokra külön ki kell adni a frissítést. A mezők tartalma a sablonban a dokumentum adatlapja alapján automatikusan kerül kitöltésre.

A diplomaterv szabványos méretű A4-es lapokra kerüljön. Az oldalak tükörmargóval készüljenek (mindenhol 2,5 cm, baloldalon 1 cm-es kötéssel). Az alapértelmezett betűkészlet a 12 pontos Times New Roman, másfeles sorközzel.

Minden oldalon – az első négy szerkezeti elem kivételével – szerepelnie kell az oldalszámnak.

A fejezeteket decimális beosztással kell ellátni. Az ábrákat a megfelelő helyre be kell illeszteni, fejezetenként decimális számmal és kifejező címmel kell ellátni. A fejezeteket decimális aláosztással számozzuk, maximálisan 3 aláosztás mélységben (pl. 2.3.4.1.). Az ábrákat, táblázatokat és képleteket célszerű fejezetenként külön számozni (pl. 2.4. ábra, 4.2 táblázat vagy képletnél (3.2)). A fejezetcímeket igazítsuk balra, a normál szövegnél viszont használjunk sorkiegyenlítést. Az ábrákat, táblázatokat és a hozzájuk tartozó címet igazítsuk középre. A cím a jelölt rész alatt helyezkedjen el.

A képeket lehetőleg rajzoló programmal készítsék el, az egyenleteket egyenlet-szerkesztő segítségével írják le.

Az irodalomjegyzék szövegközi hivatkozása történhet a Harvard-rendszerben (a szerző és az évszám megadásával) vagy sorszámozva. A teljes lista névsor szerinti sorrendben a szöveg végén szerepeljen (sorszámozott irodalmi hivatkozások esetén hivatkozási sorrendben). A szakirodalmi források címeit azonban mindig az eredeti nyelven kell megadni, esetleg zárójelben a fordítással. A listában szereplő valamennyi publikációra hivatkozni kell a szövegben. Minden publikáció a szerzők után a következő adatok szerepelnek: folyóirat cikkeknél a pontos cím, a folyóirat címe, évfolyam, szám, oldalszám tól-ig. A folyóirat címeket csak akkor rövidítsük, ha azok nagyon közismertek vagy nagyon hosszúak. Internet hivatkozások megadásakor fontos, hogy az elérési út előtt megadjuk az oldal tulajdonosát és tartalmát (mivel a link egy idő után akár elérhetetlenné is válhat), valamint az elérés időpontját.

Fontos:

* a szakdolgozat készítő/diplomatervező nyilatkozata (a jelen sablonban szereplő szövegtartalommal) kötelező előírás Karunkon, ennek hiányában a szakdolgozat/diplomaterv nem bírálható és nem védhető!
* mind a dolgozat, mind a melléklet maximálisan 15 MB méretű lehet!

Jó munkát, sikeres szakdolgozat készítést, ill. diplomatervezést kívánunk!

FeladatkiÍrás

A feladatkiírást a tanszék saját előírása szerint vagy a tanszéki adminisztrációban lehet átvenni, és a tanszéki pecséttel ellátott, a tanszékvezető által aláírt lapot kell belefűzni a leadott munkába, vagy a tanszékvezető által elektronikusan jóváhagyott feladatkiírást kell a Diplomaterv Portálról letölteni és a leadott munkába belefűzni (ezen oldal HELYETT, ez az oldal csak útmutatás). Az elektronikusan feltöltött dolgozatban már nem kell megismételni a feladatkiírást.

C:\Users\szarnyasg\Downloads\bme_logo_nagy.eps

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Készítette

Konzulens

2015

Tartalomjegyzék

[Összefoglaló 6](#_Toc417816953)

[Abstract 7](#_Toc417816954)

[1. Bevezetés 8](#_Toc417816955)

[1.1. Formázási tudnivalók 8](#_Toc417816956)

[1.1.1. Címsorok 8](#_Toc417816957)

[1.1.2. Képek 8](#_Toc417816958)

[1.1.3. Táblázatok 8](#_Toc417816959)

[1.1.4. Kódrészletek 8](#_Toc417816960)

[1.1.5. Irodalomjegyzék 9](#_Toc417816961)

[1.1.6. Margók 9](#_Toc417816962)

[2. Utolsó simítások 10](#_Toc417816963)

[3. Összefoglalás 11](#_Toc417816964)

[4. Bevezetés 12](#_Toc417816965)

[5. Az EMF-IncQuery, mint inkrementális, gráflekérdező keretrendszer 13](#_Toc417816966)

[6. A tesztelésről röviden 14](#_Toc417816967)

[7. A tesztkészlet felmérése 15](#_Toc417816968)

[8. Javaslattétel a tesztkészlet bővítésére 16](#_Toc417816969)

[9. Mások tapasztalatai grafikus felhasználói felület tesztelésével 17](#_Toc417816970)

[10. Az RCP Testing Tool 18](#_Toc417816971)

[11. Tesztesetek continous integration környezetbe való áthelyzése 19](#_Toc417816972)

[11.1. qgears-es megoldás 19](#_Toc417816973)

[11.2. RCPTT Runner és Maven Plugin használata 19](#_Toc417816974)

[11.3. Konklúzió 19](#_Toc417816975)

[Köszönetnyilvánítás 20](#_Toc417816976)

[Ábrák jegyzéke 21](#_Toc417816977)

[Táblázatok jegyzéke 22](#_Toc417816978)

[Irodalomjegyzék 23](#_Toc417816979)

[Függelék 24](#_Toc417816980)

Hallgatói nyilatkozat

Alulírott , szigorló hallgató kijelentem, hogy ezt a szakdolgozatot/diplomatervet (nem kívánt törlendő) meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2015. 04. 29.

Bátor András Joakim

# Összefoglaló

A szakdolgozat magyar nyelvű kivonata. Ez egy ½–1 oldalas magyar nyelvű összefoglaló, melynek szövege a Diplomaterv Portálra külön is feltöltésre kerül.

# Abstract

English abstract of the thesis work. This summarises the content of the thesis in 0.5–1 pages and is uploaded to the Thesis Work Portal as well.

# Bevezetés

A következő fejezet pár példán keresztül bemutatja a diplomatervekben és szakdolgozatokban szokásosan előkerülő formázások megvalósítását.

## Formázási tudnivalók

A dokumentum folyószövegéhez használjuk a **Normál** (angol Word esetén Normal) stílust.

### Címsorok

A fejezetcímek esetén a **Címsor 1–4** (Heading 1–4) stílusokat használjuk.

### Képek

A képhez használjuk a **Kép** stílust. Képaláírást a képen jobb gombbal kattintva a **Képaláírás beszúrása…** opcióval adhatunk hozzá, így az automatikusan **Képaláírás** (Caption) stílusú lesz.



1.1. ábra. Példa képaláírásra

Képek hivatkozásához jelöljük ki a képaláírásban a sorszámot (pl. „1.1.”), majd kattintsunk a **Könyvjelző** gombra, majd hozzunk létre egy könyvjelzőt (pl. „bmelogo” névvel). Ezután a **Kereszthivatkozás** gombra kattintva a **Hivatkozástípus**t állítsuk **Könyvjelző**re és válasszuk ki a **bmelogo** könyvjelzőt. Így ehhez hasonló hivatkozásokat készíthetünk: lásd az XXX. ábrán.

### Táblázatok

A dolgozatban szereplő táblázatokat az 1.1. táblázat mintájára érdemes elkészíteni.

|  |  |  |
| --- | --- | --- |
| ****Fejléc**** | ****Opció A**** | ****Opció B**** |
| 1. sor |  |  |
| 2. sor |  |  |

1.1. táblázat. Példa táblázat feliratára

### Kódrészletek

Kódrészletek beillesztése esetén használjuk a **Kód** stílust.

using System;

namespace MyApp

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Szia Világ!");

}

}

}

### Irodalomjegyzék

Az irodalomjegyzék kezelése többféleképpen is megoldható, az alábbiakban két egyszerű módszert ismertetünk.

#### Kereszthivatkozásokkal

A kereszthivatkozásokkal történő irodalomjegyzék egy megfelelően formázott felsorolás, melynek egyes elemeire (bekezdéseire) mutatnak hivatkozások. Jelen dokumentum ezt a megközelítést alkalmazza.

Az Irodalomjegyzékben szereplő hivatkozásokat **Irodalomjegyzék bejegyzés** stílussal formázzuk, a címüket pedig a **Kiemelés** stílussal emeljük ki.

A szövegbe a hivatkozásokat a **Kereszthivatkozás beszúrása** (Insert cross-reference) funkcióval helyezzük el (példa egy így beszúrt hivatkozásra: [1]), így azok később frissíthetők a hivatkozások átrendezése esetén (lásd 2. fejezet).

#### Források kezelése

A kereszthivatkozások alternatívája, hogy a hivatkozott műveket először felvesszük a szerkesztőprogram adatbázisába. Új műveket a **Hivatkozás** fülön a **Források kezelése** alatt az **Új…** gombbal vehetünk fel. A szerzőket érdemes a **Szerző** mező mellett található **Szerkesztés** gomb használatával felvenni.

Az irodalomjegyzéket az **Irodalomjegyzék** gomb alatt az **Irodalomjegyzék** opcióval szúrhatjuk be a dokumentumba. A hivatkozások stílusa a **Stílus** gomb alatt állítható be, a javasolt stílus az **IEEE**.

### Margók

Az **Oldalbeállítás** menüben ellenőrizzük a **Kötésmargó** beállítását. Amennyiben a dolgozat kétoldalas nyomtatással készül, a **Több oldal** beállításnál válasszuk a **Margók tükrözése** opciót.

# Utolsó simítások

Miután elkészültünk a dokumentációval, ne felejtsük el a következő lépéseket:

* Kereszthivatkozások frissítése: miután kijelöltük a teljes szöveget (Ctrl + A), nyomjuk meg az F9 billentyűt, és a Word frissíti az összes kereszthivatkozást. Ilyenkor ellenőrizzük, hogy nem jelent-e meg valahol a „Hiba! A könyvjelző nem létezik.” szöveg.
* Dokumentum tulajdonságok megadása: a dokumentumhoz tartozó metaadatok kitöltése (szerző, cím, kulcsszavak stb.). Ez Word 2013 alatt a **Fájl** | **Információ** | **Tulajdonságok** | **Dokumentumpanel megjelenítése** gombra kattintva érhető el.
* Kinézet ellenőrzése PDF-ben: a dokumentum elkészítése után feltétlenül ellenőrizzük a kapott PDF dokumentumot is.

# Összefoglalás

A diplomaterv összefoglalása.

# Bevezetés

Az Eclipse platform és az erre épülő alkalmazások meghatározó szerepet foglalnak el a szoftverfejlesztés iparágban. Az úgynevezett nyílt-forráskódú szoftverek nagy része is erre a platformra épül. Szakdolgozatom témája is egy ilyen alkalmazás tesztelése.

A modellvezérelt rendszer- és szoftverfejlesztés napjaink központi jelentőségű módszertana az összetett, minőségi és biztonsági szempontból kritikus beágyazott rendszerek valamint a komplex szolgáltatásorientált szoftverrendszerek területén. A módszertan központi fontosságú komponensei a modelleken értelmezett lekérdezések, melyek kulcsszerepet játszanak a nyelvközi átalakítások, forráskódszintézis, tesztesetek, dokumentációrészletek automatikus generálása során is.

## EMF-IncQuery, a gráflekérdező keretrendszer

A Méréstechnika és Információs Rendszerek Tanszék részvételével fejlesztett EMF-IncQuery inkrementális gráflekérdező keretrendszer fontos eleme az iparban széles körben használt Eclipse Modeling platformnak, és számos nyílt forrású, illetve kereskedelmi modellezőeszköz részeként a modellvezérelt eszközfejlesztés egyik alaptechnológiájának számít.

A keretrendszer segítségével lekérdezéseket fogalmazhatunk meg EMF modellek felett, és ezek végrehajtása automatikusan megtörténik, anélkül, hogy valamilyen magas szintű programnyelvet – mint például a Java – kellene használnunk.

Az EMF-IncQuery fő funkciói:

* Xtext alapú nyelv használata a lekérdezések megfogalmazására
* Lekérdezések optimalizálása: a példány modell egyszeri teljes lekérdezése, utána a változások figyelése (inkrementális jelleg)
* IncQuery integrálása más alkalmazásokba a különböző API-k használatával

# A tesztelésről röviden

Mivel a feladatom az EMF-IncQuery eszköz integrációs tesztelése volt, fontos volt felderíteni az eddigi tesztkészletet, ami a fejlesztés során létrejött. Ahhoz, hogy ezeket a teszteket pontosan lehessen kategorizálni, tudnunk kell, hogy a szoftver tesztelésnek milyen szintjei lehetnek:

* komponensteszt,
* integrációs teszt,
* rendszerteszt,
* átvételi teszt.

A komponensteszt csak a rendszer egy komponensét teszteli önmagában. Az integrációs teszt kettő vagy több komponens együttműködési tesztje. A rendszerteszt az egész rendszert, tehát minden komponenst együtt, teszteli. Ez első három teszt szintet együttesen fejlesztői tesztnek hívjuk, mert ezeket a fejlesztő cég alkalmazottai vagy megbízottjai végzik. Az átvételi teszt során a felhasználók a kész rendszert tesztelik. Ezek általában időrendben is így követik egymást.

A komponensteszt a rendszer önálló részeit teszteli általában a forráskód ismeretében (fehér dobozos tesztelés). Gyakori fajtái:

* unit-teszt,
* modulteszt.

A unit-teszt, vagy más néven egységteszt, a metódusokat teszteli. Adott paraméterekre ismerjük a metódus visszatérési értékét (vagy mellékhatását). A unit-teszt megvizsgálja, hogy a tényleges visszatérési érték megegyezik-e az elvárttal. Ha igen, sikeres a teszt, egyébként sikertelen. Elvárás, hogy magának a unit-tesztnek ne legyen mellékhatása.

A unit-tesztelést minden fejlett programozási környezet (integrated development environment, IDE) támogatja, azaz egyszerű ilyen teszteket írni. A jelentőségüket az adja, hogy a futtatásukat is támogatják, így egy változtatás után csak lefuttatjuk az összes unit-tesztet, ezzel biztosítjuk magunkat, hogy a változás nem okozott hibát. Ezt nevezzük regressziós tesztnek.

A modulteszt általában a modul nem-funkcionális tulajdonságát teszteli, pl. sebességét, vagy, hogy van-e memóriaszivárgás (memory lake), van-e szűk keresztmetszet (bottleneck).

Az integrációs teszt során a komponensek közti interfészeket, az operációs rendszer és a rendszer közti interfészt, illetve más rendszerek felé nyújtott interfészeket tesztelik. Az integrációs teszt legismertebb típusai:

* Komponens – komponens integrációs teszt: A komponensek közötti kölcsönhatások tesztje a komponensteszt után.
* Rendszer – rendszer integrációs teszt: A rendszer és más rendszerek közötti kölcsönhatásokat tesztje a rendszerteszt után.

Az integrációs teszt az összeillesztés során keletkező hibákat keresi. Mivel a részeket más-más programozók, csapatok fejlesztették, ezért az elégtelen kommunikációból súlyos hibák keletkezhetnek. Gyakori hiba, hogy az egyik programozó valamit feltételez (pl. a metódus csak pozitív számokat kap a paraméterében), amiről a másik nem tud (és meghívja a metódust egy negatív értékkel). Ezek a hibák kontraktus alapú tervezéssel (design by contract) elkerülhetőek.

# A tesztkészlet felderítése

Az EMF-IncQuery-hez számos teszt készült az évek során, de ezek szétszórva találhatóak meg az interneten, különböző repository-kban, és különböző példány modelleket használnak a teszteléshez. Természetesen a tesztelés szintjében is különböznek ezek a tesztek. Tehát első dolgom volt a már meglévő tesztek katalogizálása különböző szempontok alapján:

* Teszt projekt neve
* Használt metamodell
* Teszt típusa
* Tesztelés tárgya
* Fedettség mérés
* Tartalmazó repository
* Continous Integration-ben használva vannak-e

A következőkben ezeket a teszteket fogom elemezni a megadott szempontok alapján. Az elemzés végén egy rövid összefoglaló táblázat is található a gyorsabb áttekintés érdekében.

## incquery-core-repo

**patternlanguage.emf.tests**

ParserTest: A patternlanguage-ben használható elemek definiálása, és azoknak a parse-olásának tesztelése.

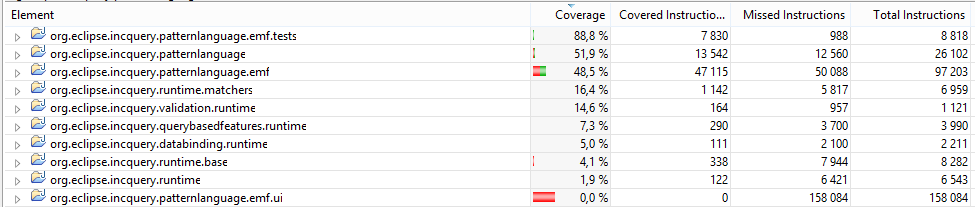
patternlanguage.emf.tests.\*:

A patternlanguage nyelvi elemeinek tesztelése:

* Annotáció
* Kompozíció
* Importálás
* Típus

Fehér dobozos tesztelés, a legtöbb helyen csak a hiba nélküli feldolgozás, vagy a megfelelő hiba kapásának ellenőrzése. A megfelelő helyeken a kapott értékek ellenőrzése is.

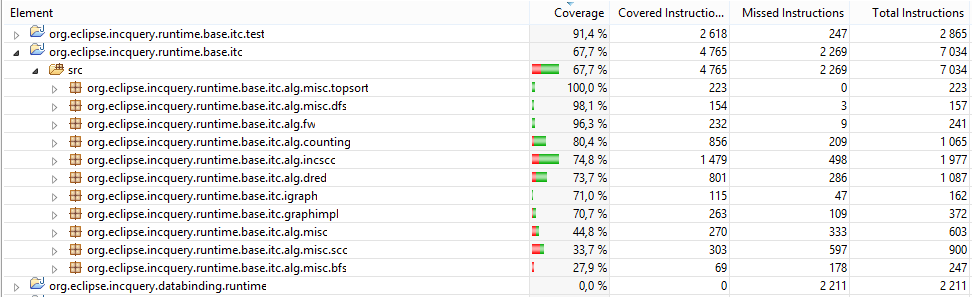
Kód-fedettség:



**runtime.base.itc.test**

Gráf implementáció tesztelése, valamint a bejárási algoritmusok tesztelése, mint például a DFS algoritmus vagy a FloydWarshall algoritmus.

Kód-fedettség:



**runtime.tests**

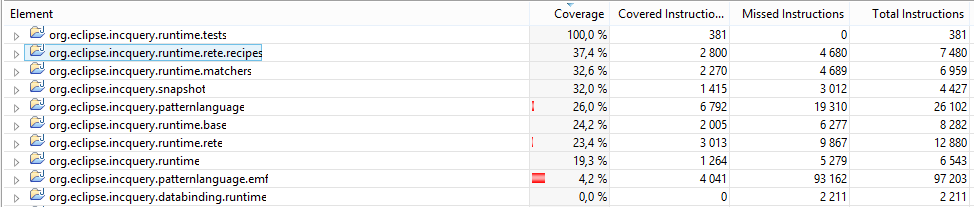
FunctionalDependencyHelperTest:

Matcher tesztelése, funkcionális függőségek lezártjának vizsgálatának a tesztelése.

EverythingDynamicTest:

Dinamikus metamodell-en dolgozik, ami a tesztesetben kreálódik. Utána létrehoz egy példánymodellt is, és azon hajt végre egy lekérdezést, amin asszertál. Ez egy integrációs teszt. Az eiqsnapshot.ecore metamodellt használja.

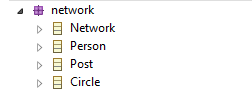
Kód-fedettség:



## eiq-examples-repo:

**network.tests**

Runtime IncQuery kell hozzá, egyetlen teszteset. A lekérdezendő network metamodellje:



Különböző méretű modellek állíthatók be a lekérdezéshez (small, medium, large). Kisebb módosításra volt szükség a CollectionsFactory és CollectionsFramework osztályok eléréséhez.

Teljesítmény teszt:

Medium model esetén:

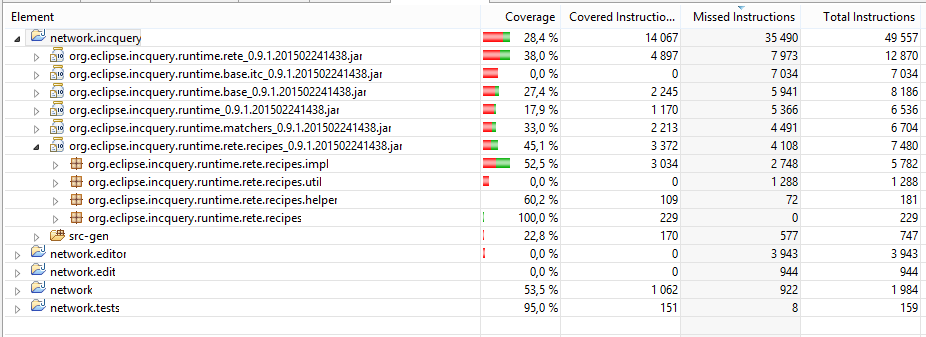
[Java] Matches: 9082

[Java] Elapsed seconds: 3.102905032

Large model esetén:

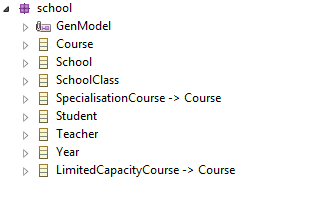
java.lang.OutOfMemoryError: GC overhead limit exceeded

Kód-fedettség:



**school.tests**

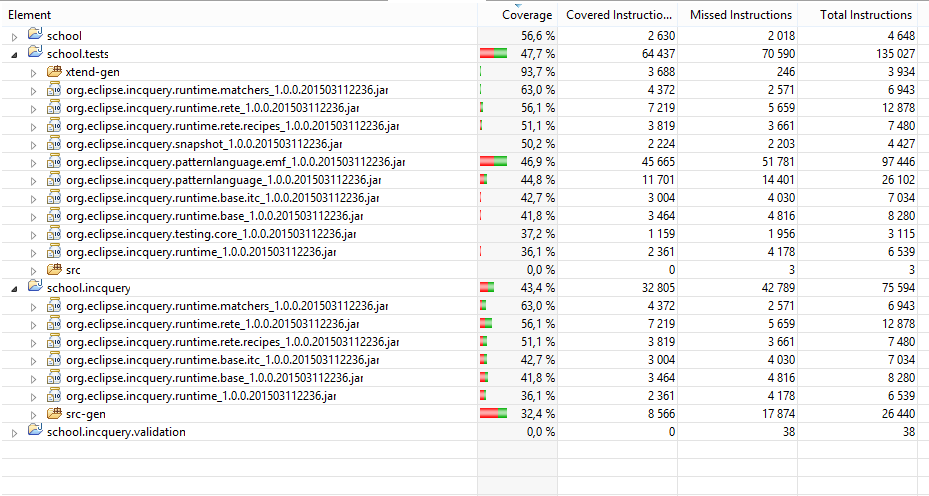
Különböző fajta tesztek a School metamodellen:



Tesztek:

* API-Matcher teszt
* Basic query teszt
* Lekérdezés közben modell manipulálás tesztelése
* Rekurzív lekérdezés
* Snapshot tesztelés

Kód-fedettség:



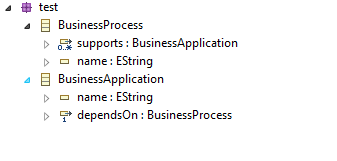
**bpmn.tests**

Erroros eiq fájlok, import <http://stp.eclipse.org/bpmn> nem oldható fel.

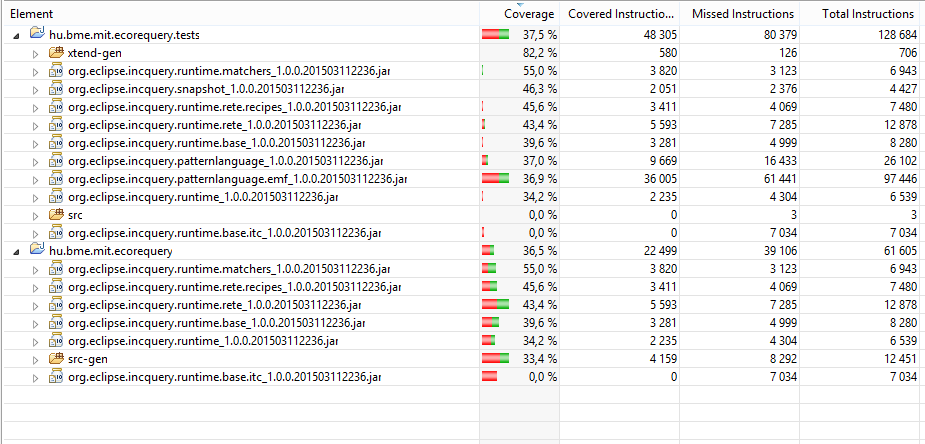
**hu.bme.mit.ecorequery.tests**

A School példa egyszerű lekérdezéseinek tesztelése az ecore modellen. Valamint a modell változtatás tesztelése.

Modell (+ Ecore):



Kód-fedettség:



**derivedTest**

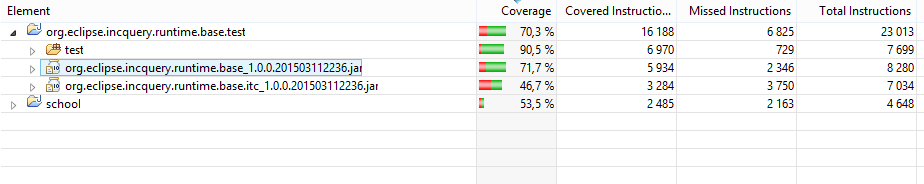
Csak IncQuery 0.6-tal kompatibilis.

**runtime.base.test**

A különböző EMF típusokhoz getterek tesztelése dinamikus EMF modellekhez is (AttributeValue, DataType, FeatureTest, InstanceTest, ListenerTest …)

A School.ecore metamodellen kívül használva van ezeknek két példánymodellje is, a School1.school és a School2.school.

Kód-fedettség:

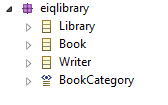


## incquery-examples-repo

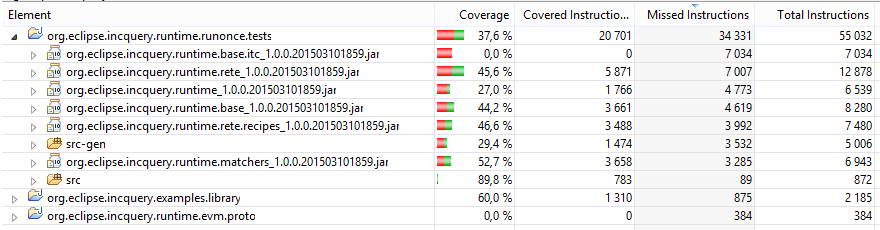
**runtime.runonce.tests**

Test cases that run different kind of derived features in run-once engine.

Lekérdezett modell:



Kód-fedettség:



**examples.bpm.tests**

Ez az eiq-examples repo-ban levő bpmn nevű tesztnek a felfrissített verziója, vagyis ami működik az 1.0-ás verziójú IncQuery-vel is.

A használt metamodellek eben az esetben a következők:

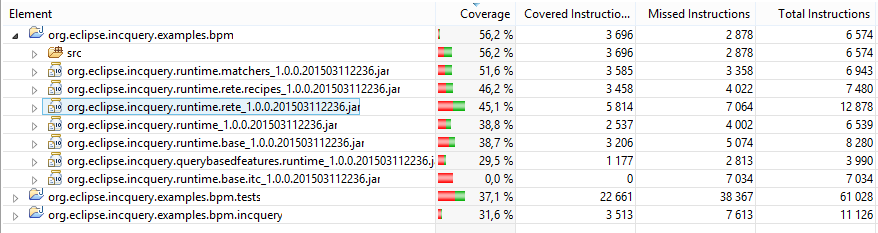
* derivedModel.ecore
* integrated.ecore
* integration.ecore
* operation.ecore
* process.ecore
* system.ecore

Valamint a lekérdezésekhez használt példánymodellek:

* Simple.operation
* Simple.process
* Simple.system

Ez is egy integrációs teszt, lényegében a School példa BPMN modell megfelelője. Bár itt kevesebb dolog van tesztelve. A Matcher használata lekérdezésekhez. Ezek a tesztek főként a QueryBasedFeatureTest-ben találhatóak. A RuntimeListenerTest pedig az IncQueryEngine alapvető funkcióinak tesztelése található.

Kód-fedettség:



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMF-IncQuery tesztkatalógus (1/2) | | | | | | |
| Teszt-projekt neve | **Használt metamodell** | **Teszt típusa** | **Tesztelés tárgya** | **Fedettség mérés (legjobban fedett komponensek)** | **Tartalmazó repository** | **CI[[1]](#footnote-1)** |
| patternlanguage.emf.tests | - | komponensteszt  (unit-teszt) | Patternlanguage nyelvi elemeinek tesztelése, parse-olás tesztelése. | patternlanguage (50%)  patternlanguage.emf (50%) | incquery-core-repo | Igen |
| runtime.base.itc.test | - | komponensteszt  (unit-teszt) | Gráf implementáció tesztelése, valamint a bejárási algoritmusok tesztelése. | runtime.base.itc (67%) | incquery-core-repo | Igen |
| runtime.tests | dinamikusan létrehozott | komponens integrációs teszt | Matcher és funkcionális függőségek fedésének tesztelése. | runtime (30%),  patternlanguage (30%),  snapshot (30%) | incquery-core-repo | Igen |
| network.tests | network.ecore | komponensteszt (modul-teszt) | Különböző méretű példánymodelleken futás idejének vizsgálata. | runtime (30%) | eiq-examples-repo | Nem |
| school.tests | school.ecore | komponens integrációs teszt | Különböző lekérdezések és framework tesztelés a School példával | runtime (50%),  patternlanguage (50%),  snapshot (50%) | eiq-examples-repo | Igen |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| EMF-IncQuery tesztkatalógus (2/2) | | | | | | |
| Teszt-projekt neve | **Használt metamodell** | **Teszt típusa** | **Tesztelés tárgya** | **Fedettség mérés (legjobban fedett komponensek)** | **Tartalmazó repository** | **CI[[2]](#footnote-2)** |
| ecorequery.tests | ecore.ecore | komponens integrációs teszt | Ecore.ecore modellen végzett lekérdezések | runtime (40%),  patternlanguage (40%) | eiq-examples-repo | Igen |
| bpmn.tests | csak példánymodell | elavult | elavult | elavult | eiq-examples-repo | Nem |
| derivedTest | derivedModel.ecore | elavult | elavult | elavult | eiq-examples-repo | Nem |
| runtime.base.test | school.ecore (school1.school és school2.school) | komponensteszt  (unit-teszt) | A különböző EMF típusokhoz getterek tesztelése dinamikus EMF modellekhez is. | runtime.base (70%),  runtime.base.itc (50%) | eiq-examples-repo | Nem |
| runtime.runonce.tests | eiqlibrary.ecore | komponens integrációs teszt | Run-once engine használata a különböző funkciók teszteléséhez. | runtime.matchers (50%),  runtime.rete (45%),  runtime.base (45%) | examples-repo | Nem |
| bpm.tests | derivedModel.ecore  integrated.ecore  integration.ecore  operation.ecore  process.ecore  system.ecore | komponensteszt  (unit-teszt) | BPMN modellen lekérdezések és IncQueryEngine funkcióinak tesztelése. | runtime.matchers (50%),  runtime.rete (45%) | examples-repo | Igen |

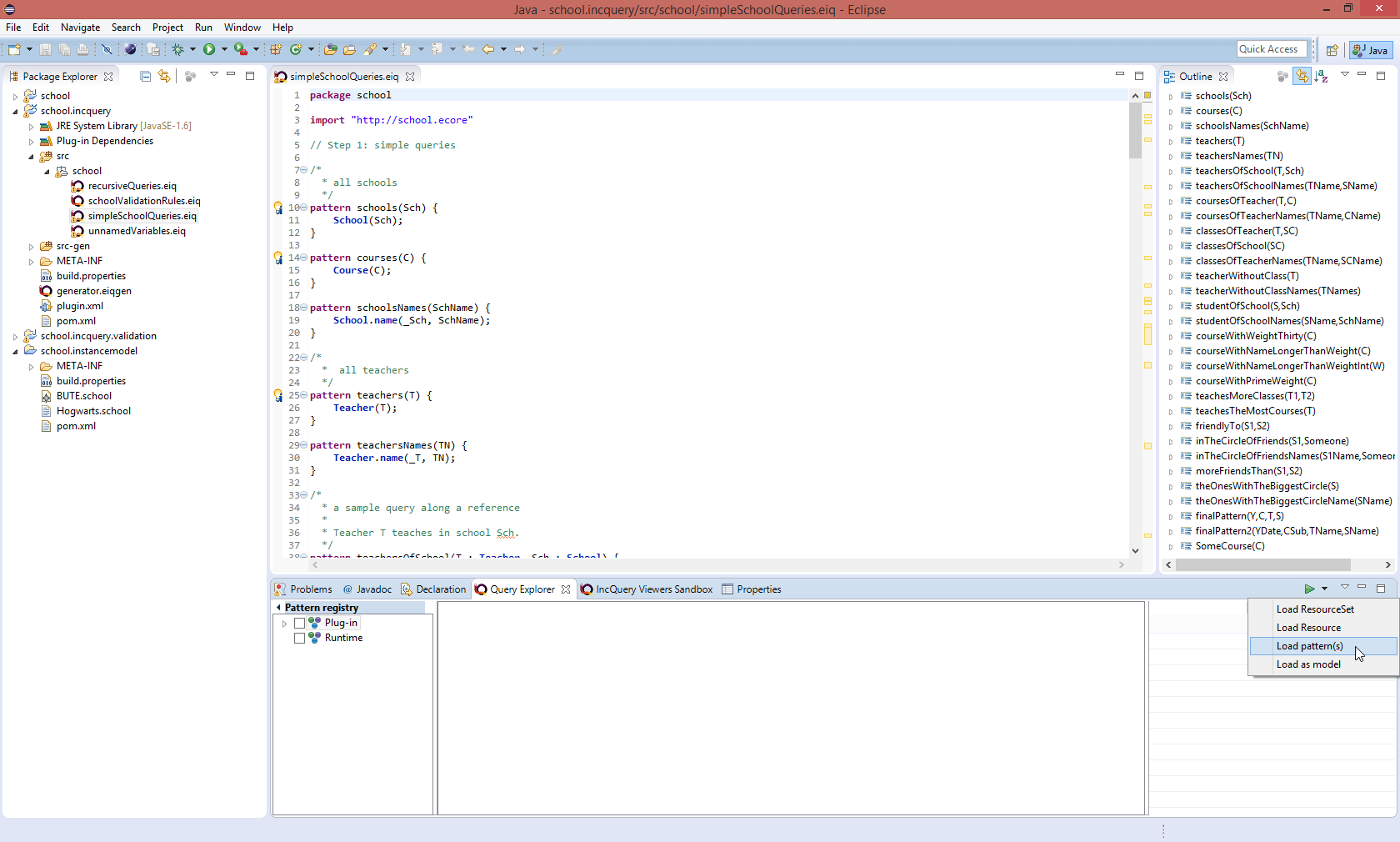
# Javaslattétel a tesztkészlet bővítésére

Mint az a korábbiakban olvasható volt, az eddig létrehozott tesztek főként a belső működés vizsgálatára készültek. A felhasználói felület tesztelésével egyik teszteset sem foglalkozik. A fejlesztők elmondása szerint, azt eddig csak manuálisan kattintgatva tesztelték időről időre, vagy éppen az adott komponens fejlesztése során.

Ezért felmerült az felhasználói felület tesztelésének igénye. Ehhez először is össze kellett szedni az EMF-IncQuery grafikus felhasználói felületének elemeit, és a use-case-eket. Ez lényegében a lehetséges felhasználói interakciókat jelentette. A következőkben ezeknek a felsorolása következik.

## Query Editor

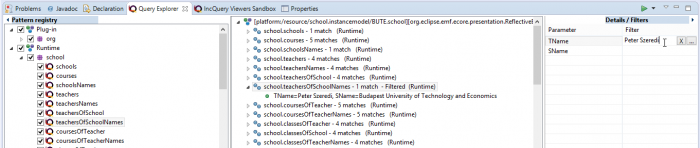
A Query Editor egy Xtex alapú szerkesztő, amivel EMF-IncQuery lekérdezéseket szerkeszthetünk. Ez a szerkesztő rengeteg fejlesztést segítő megoldást tartalmaz, mint például az automatikus kód kiegészítés, amit már más fejlesztő eszközökből is megismerhettünk. Mivel ennek a szerkesztőnek a vizsgálata GUI tesztelő eszközökkel kényelmetlen lenne, a mintanyelv egyes elemei pedig már más tesztek által le vannak fedve, ezért a Query Editor tesztelése nem fő célja a tesztesetek bővítésének.



2. ábra - A Query Editor kinézete

## Query Explorer [5]

A Query Explorer az EMF-IncQuery legfontosabb és legösszetettebb GUI komponense. Ennek segítségével tudunk lekérdezéseket végrehajtani EMF példánymodelleken. Együttműködik EMF alapú szerkesztőkkel, mint például az alap EMF fa szerkesztő, GMF vagy éppen Graphiti szerkesztők. A Query Explorer ezen kívül képes a lekérdezések utólagos feldolgozására is, például a találati lista szűrésére vagy a részletek megjelenítésére.

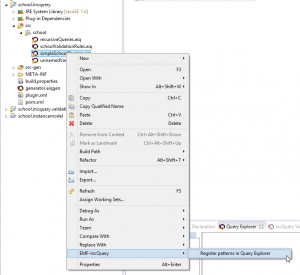


3. ábra - A Query Explorer

A sokoldalú funkcionalitásnak köszönhetően ez a komponens több kisebb komponensből épül fel. A következőkben ezek bemutatása következik.

### Kontextus menü (EMF-IncQuery)

Egy mintákat tartalmazó .eiq fájlra jobb gombbal kattintva a kontextus függő menüben a szokványos menüelemek, - mint a „Run As, Debud As” – megtalálhatjuk az EMF-IncQuery -> Register patterns in Query Explorer menüt is, aminek segítségével mintáinkat felvehetjük a Pattern Registry-be. Ez szolgál ugyanis a mintáink menedzselésére, hogy éppen melyiket szeretnénk a lekérdezéshez használni.

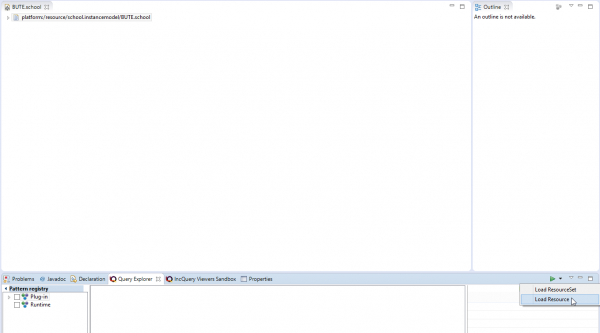


4. ábra - Pattern regisztrálása a Query Explorer-ben

### „Magic Green Button”

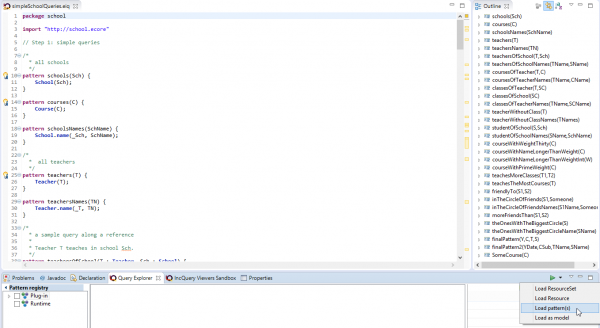
A „Magic Green Button” működése szintén kontextusfüggő. Miután a példánymodellünket megnyitottuk a Sample Reflective Ecore Model Editor-al a Query Explorer zöld gombja aktívvá válik és használatra kész. Szöveges szerkesztő esetén ez nem történik meg. A workbench pillanatnyi állásától függően ezután a következőkre van lehetőségünk:

* Load ResourceSet: Ezzel betölthetjük az egész példánymodellt, ami a szerkesztőben megjelenik (ez az alapértelmezett a zöld gomb megnyomása esetén).
* Load Resource: Csak a kiválasztott elem töltődik be, a példánymodell részhalmaza.



5. ábra - "Magic Green Button" állapota megnyitott példánymodell esetén

* Load pattern(s): Amennyiben egy .eiq fájl van megnyitva, lehetőség van itt is felvenni a benne levő mintákat a Pattern Registry-be.
* Load as model: Lehetőség van .eiq fájlunk példánymodellként való betöltésére is, hogy utána ezen végezhessünk lekérdezéseket. Ez főleg az EMF-IncQuery-t fejlesztőknek hasznos.

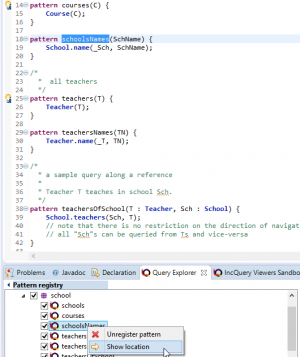


6. ábra - "Magic Green Button" állapota megnyitott .eiq fájl esetén

### Pattern registry

A Query Explorer bal oldalán elhelyezkedő Pattern Registry a betöltött mintákat mutatja, a View menu-ben beállított package hierarchiának megfelelően (lásd későbbi fejezet). Főbb funkció a következők:

* Betöltött vagy betölthető plugin és runtime patterndefiníciók mutatása
* Unregister pattern: pattern kiregisztrálása a Pattern registry-ből
  + Result Viewer automatikus frissítése ennek megfelelően
  + Plugin pattern-öket nem lehet kiregisztrálni
* Show location: pattern definiálásnak helyére ugrás
* Chekbox filtering: a jelölőnégyzetek használatával ideiglenesen kivehetjük a találati listából az egyes lekérdezéseket



7. ábra - A pattern registry-ből elérhető funkciók

### Result viewer

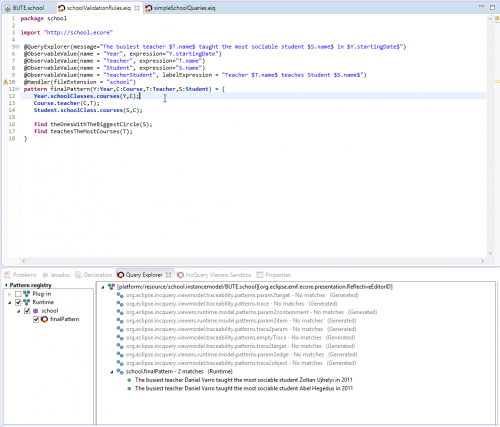
Ha legalább egy lekérdezés és egy példánymodell be van töltve, akkor a Query Explorer középső részén a lekérdezések eredményei egy fa struktúrában megjelennek. A legfelső szintű elem jelöli, hogy melyik példánymodell lett betöltve, és ez alatt találhatóak a lekérdezésekre a találatok.

A Result Viewer automatikusan frissül, amennyiben az alábbi komponensek valamelyikében valami változás történt:

* Pattern Registry
* Példánymodell
* Details/filters szekció (lásd későbbi fejezetben)

Az EMF-IncQuery lehetőséget nyújt a lekérdezések megjelenített szövegének testreszabására, ehhez mindössze a következő 2 annotációt kell használni a Query Editor-ban[6]:

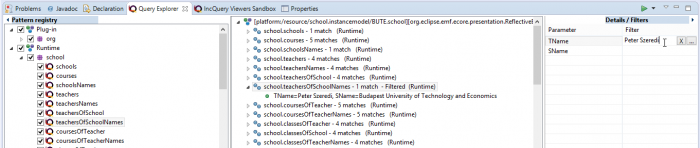
* @QueryExplorer: ennek segítségével felparaméterezhetjük a lekérdezés szövegét, ahogy az a képen is látszik
* @ObservableValue: a details panelben való megjelenés paraméterezhető fel vele



8. ábra - EMF-IncQuery adatkötésre példa

### Details/Filters

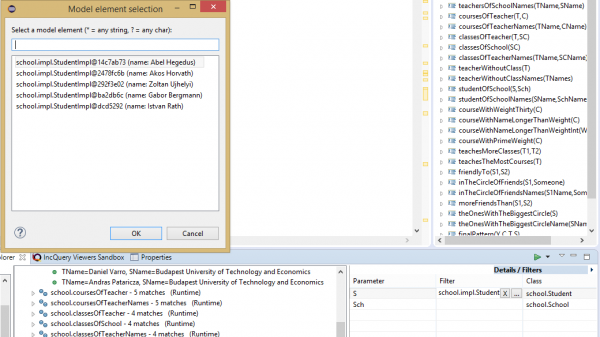
Ennek a nézetnek a tartalma a kiválasztott elemtől függ. Ha a Result Viewer-ben a lekérdezésnek a nevére kattintunk, akkor szűrőt állíthatunk be a találatokra. Amennyiben viszont egy szimpla találatra kattintunk, akkor pedig a találathoz tartozó egyes értékeket tekinthetjük meg ebben a nézetben.



9. ábra - Szűrési feltétel megadása a Details/Filters-ben

További funkciók:

* A szűrés beállításánál nem csupán szöveges megadásra van lehetőség, a „…” gombra kattintva a lehetséges szűrési értékeket kapjuk meg egy felugró ablakban, és ezek közül választhatunk.
* Természetesen itt is minden egyes változás azonnali hatással van a Result Viewer-re, a szűrési feltételek azonnal megjelennek,

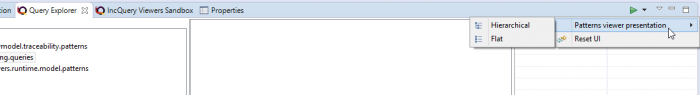


10. ábra - Felugró ablak a szűrési feltétel választásához a Details/filters nézetnél

### View menu

A képen látható, hogy a View menu hol helyezkedik el, segítségével a következőket tehetjük meg:

* Beállíthatjuk a Pattern registry package hierarchiáját (flat vagy hierarchical)
* Kitörölhetünk mindent a Query Explorer-ből. Ez hasznos abban az esetben, ha inkonzisztens állapotba kerülne valaminek a következtében.

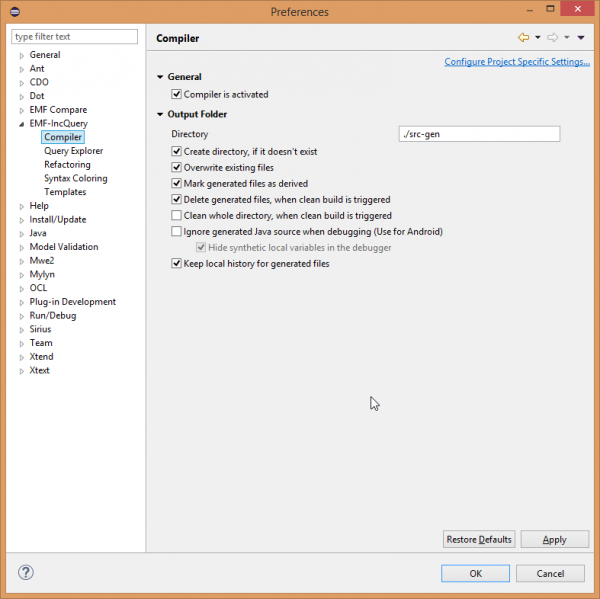


11. ábra - Lehetséges beállítások a View menu-ben

## Preferences

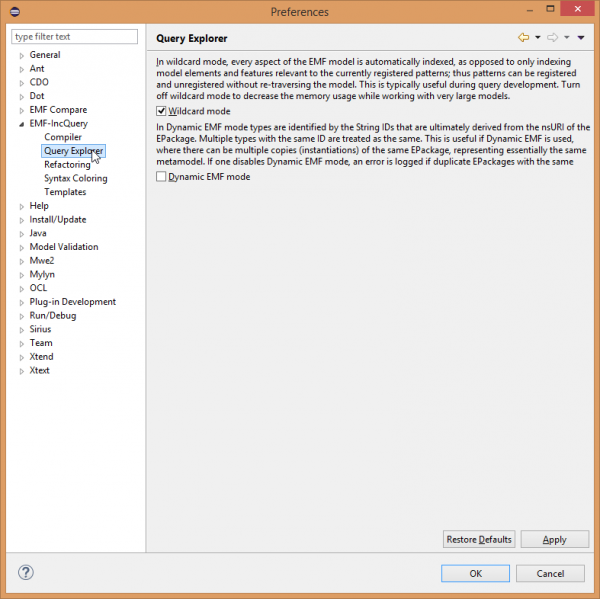
Mint szinte minden Eclipse-be épülő alkalmazás, úgy az EMF-IncQuery is rendelkezik beállítási lehetőségekkel, ami a Window -> Preferences menüpontból érhető el. Itt lehetőségünk van globális vagy éppen csak projekt specifikus beállításokat megtennünk.

* Projekt specifikus beállítások:
  + Compiler beállításai
    - Általános: itt lehet például ideiglenes kikapcsolni a compiler-t ha gyorsítani akarunk
  + Kimeneti mappa: A generált kóddal kapcsolatosan állíthatunk be dolgokat



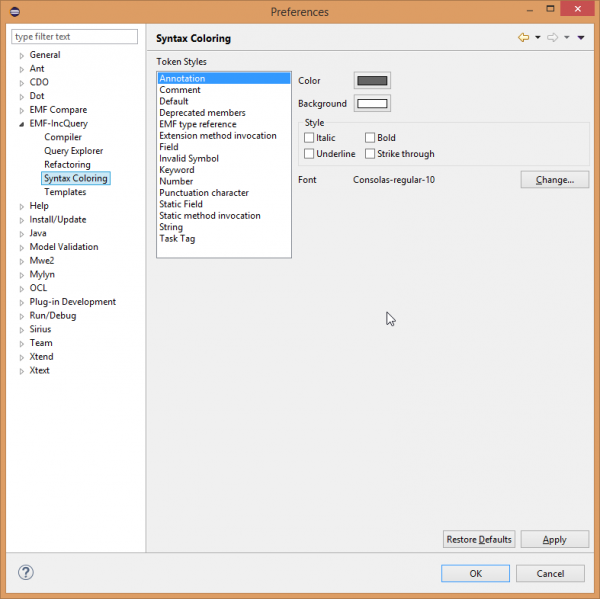
12. ábra - Compiler-re vonatkozó beállítások

* Globális beállítások
  + Query Explorer
    - Wildcard mode ki/be kapcsolása: a lekérdezés indexelésével kapcsolatos beállítás
    - Dynamic EMF mode ki/be kapcsolása: ugyannak a metamodellnek különböző példánymodellei esetén lehet hasznos



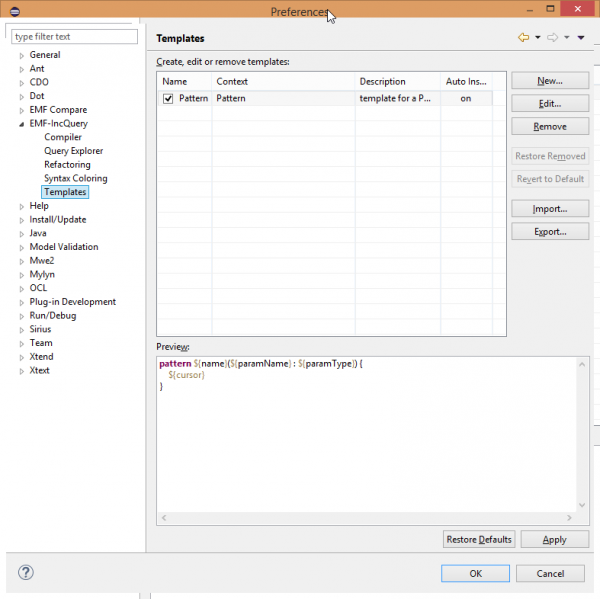
13. ábra - Query Explorer-rel kapcsolatos beállítások

* + Query Editor
    - Refactoring: Erőforrások elmentésével és átnevezésével kapcsolatos beállítások
    - Syntax Coloring: A Query Editor formázási stílusát állíthatjuk be, mint például:
      * Szín
      * Háttér
      * Stílus
      * Betűtípus



14. ábra - Syntax Coloring beállításai

* + - Templates: előre beállított stílusok
      * Kreálás, szerkesztés, törlés
      * Export, import
      * Preview (előnézeti kép az aktuális beállításoknak)



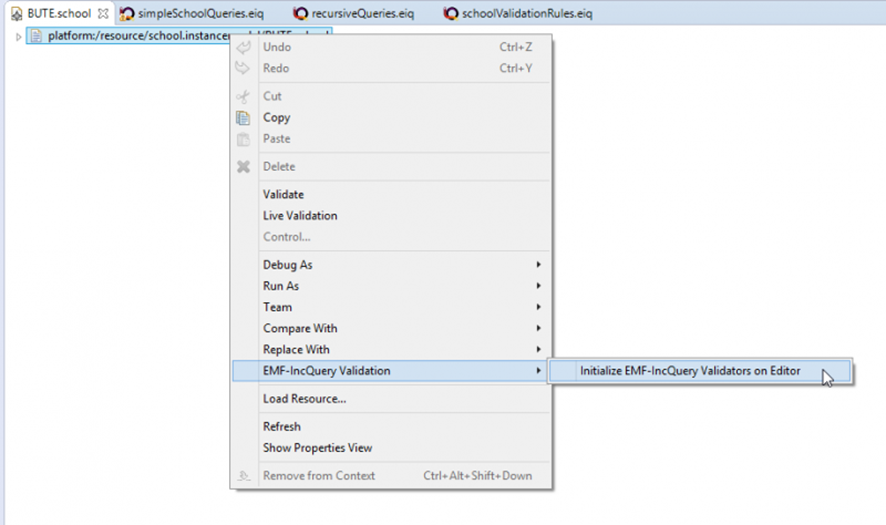
15. ábra - Templates beállításai

## Validation framework

Az EMF-IncQuery pattern language-e validációs megoldásokat is tartalmaz, ami a példánymodell folyamatos ellenőrzését jelenti. Hiba esetén pedig error-ok automatikus megjelenítését az Eclipse Problems View-ban. Ehhez a @Constraint annotációt kell használnunk a Query Editor-ban. Különböző paramétereket használhatunk a kívánt működés elérésére. Például állíthatjuk a severity-t „info”-ra, „warning”-ra vagy esetleg „error”-ra, és ennek megfelelő lesz az eredmény is.

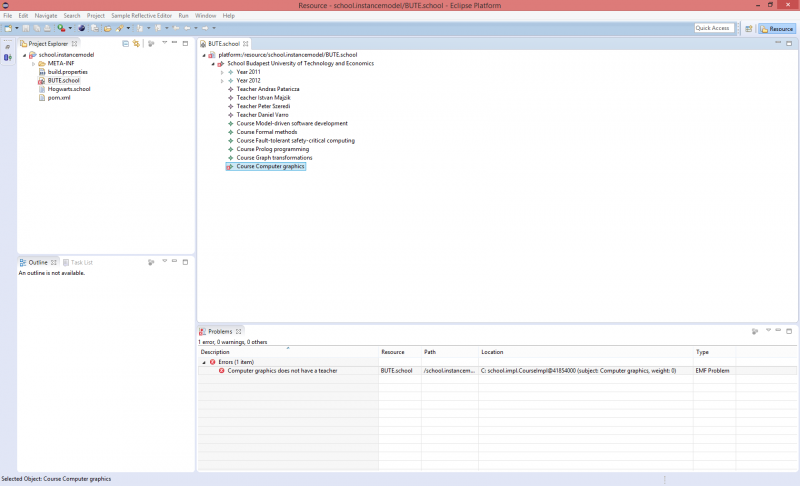
A következőkben a használatát fogom bemutatni a validation framework-nek.

1. Először is, inicializálni kell a validátorokat a példánymodellen. Ez a képen látható menüből érhető el. Ennek következtében létrejön egy .validation projekt is.



16. ábra - EMF-IncQuery validátorok inicializálása

1. Ezután el kell indítanunk a pattern-eket tartalmazó .incquery és .incquery.validation projekteket egy Eclipse alkalmazásként.
   1. az itteni változtatások a példánymodellen automatikusan megjelennek a Problems view-ban
   2. a példánymodell-beli elemre navigálhatunk (Jobb gomb -> „Go to”)
   3. különböző adatokat érhetünk el a változásról:
      1. leírás
      2. erőforrás
      3. típus
      4. hely
      5. elérési útvonal



17. ábra - Példánymodell változása esetén értesítés

## Viewers

Az EMF-IncQuery Viewer célja az egyes lekérdezések eredményeinek felhasználóbarátabb megjelenítése ezzel is segítve a modellvezérelt szoftverfejlesztést.

A megjelenítés alapjába véve különböző JFace Viewer-ekkel történik, mint a ListViewer és TreeViewer. Külön telepíthető a GEF4 Zest alapú GraphViewer is, amivel, mint a neve is mutatja gráfként is megtekinthetjük az eredményeket.

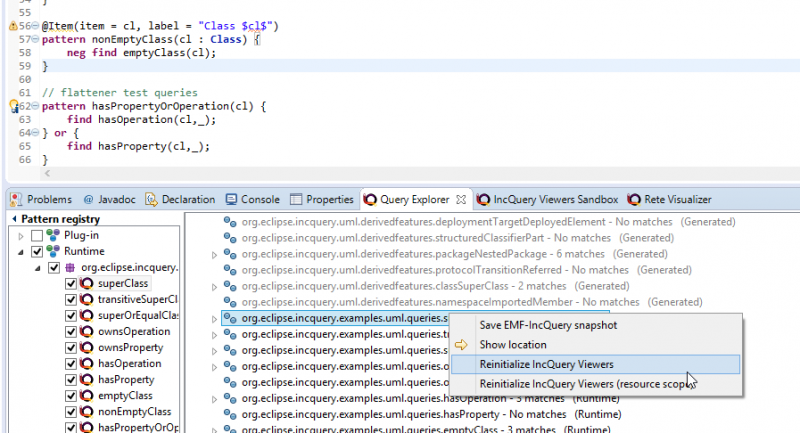
Ahhoz, hogy az ilyenfajta megjelenítés lehetséges legyen, a következő annotációkra van szükség a lekérdezések definiálásánál:

* @Item: a megjelenítendő elemeket írja le
* @ContainsItem: Item-ek közötti birtoklási viszony leírására
* @Edge: Item-ek közötti kapcsolatok leírására
* @Format: plusz formázási információ adható meg ezzel

### Viewers Sandbox

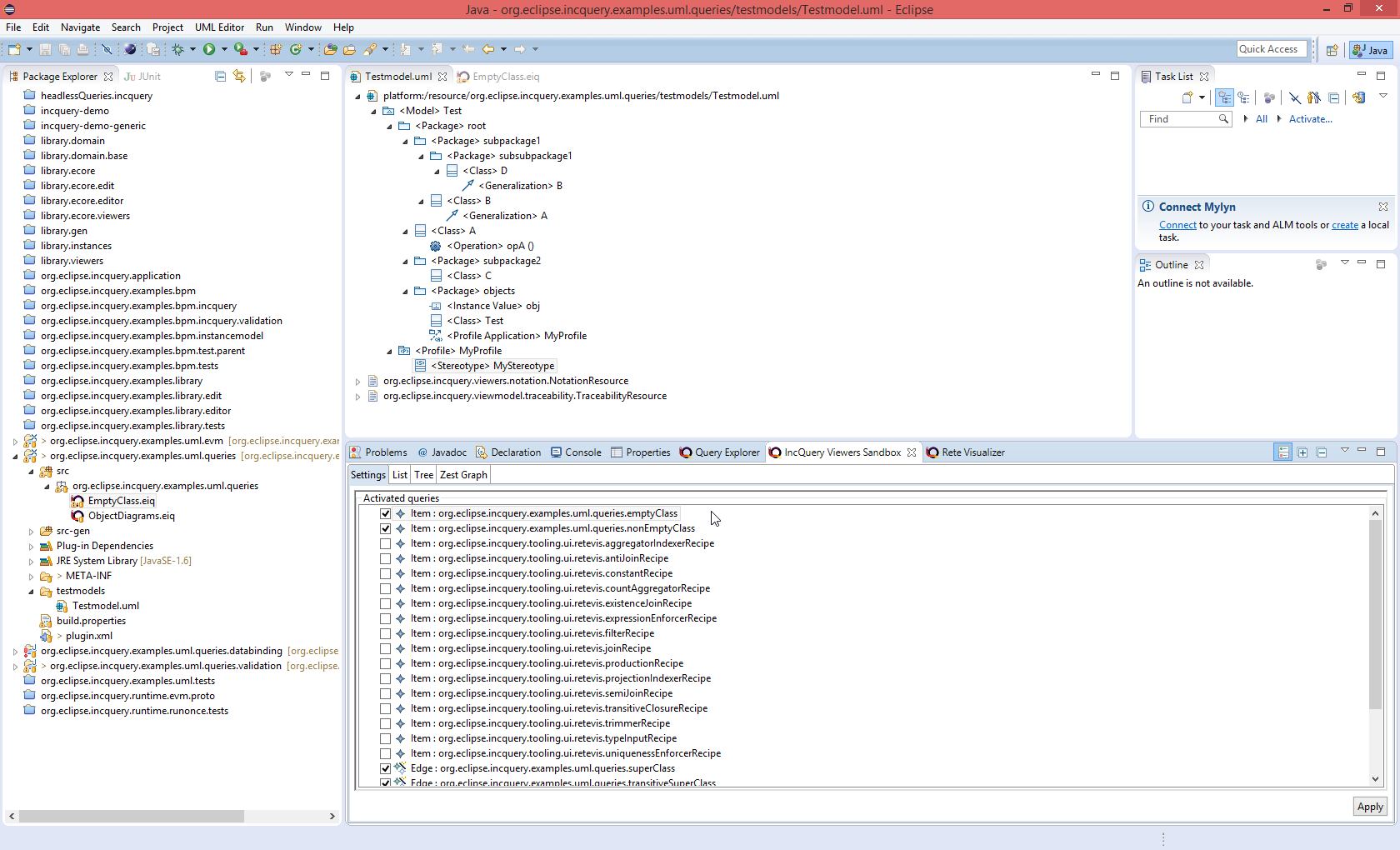
A következőkben a Viewers használatát fogom bemutatni:

* Először is a Query Explorer-ben inicializálni kell az IncQuery Viewers-t. Ehhez kell, hogy be legyen töltve legalább egy .eiq fájl és egy példánymodell. Majd a képen látható módon tudunk inicializálni:



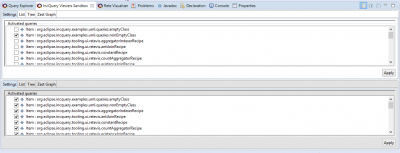
18. ábra - EMF-IncQuery Viewers inicializálása

* Ezután kezdődhet meg a nézetek konfigurálása:
  + kiválaszthatjuk, hogy melyik lekérdezések eredményei jelenjenek meg a viewer-eken
    - a nézetek szinkronizálódnak az „Apply” gombra kattintva



19. ábra - Lekérdezések kiválasztása a megjelenítéshez

* + létrehozhatunk új komponenseket
  + bezárhatunk komponenseket
  + válthatunk horizontális vagy vertikális megjelenítés között

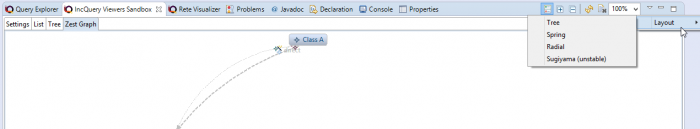


20. ábra - Horizontális és vertikális megjelenítés

Az elérhető nézetek tehát a következő képeken tekinthetőek meg:

|  |  |  |
| --- | --- | --- |
| List.png  21. ábra - ListViewer | Tree.png  22. ábra - TreeViewer | Zest.png  23. ábra - GraphViewer |

A GraphViewer-nél még további megjelenítési nézetek is elérhetőek. Ezek csak itt érhetőek el, különböző gráftípusokat jelenteken az elnevezések:

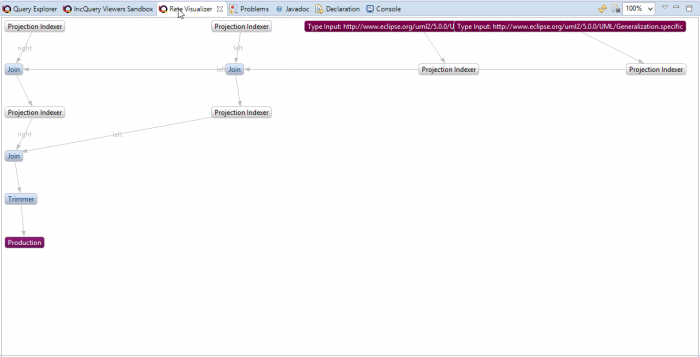


## Rete Visualizer

A Rete Visualizer-rel lehetőség nyílik a lekérdezésről plusz információk kinyerésére, megnézhetjük a query-nk Rete hálóját. A csomópontok hasznos információkkal rendelkeznek, mint például a találatok száma.

Használata:

* Példánymodell és lekérdezéseket tartalmazó .eiq fájl betöltése
* Query Explorer-ben megjeleníteni kívánt pattern kiválasztása
* A Rete háló megváltoztatásához a pattern-t ki kell regisztrálni, majd újra betölteni a „Magic Green Button”-el



24. ábra - Rete Visualizer által megjelenített háló

Ebben a nézetben is találhatunk számos beállítási lehetőséget a nézet jobb felső részében:

* Gráf frissítése
* Gráf törlése
* Zoomolás
* Layout menu-ben a kinézetet állíthatjuk be, amik a következők lehetnek:
  + Tree
  + SpaceTree
  + Spring
  + Radial
  + Sugiyama

## Testing framework

Az EMF-IncQuery Testing framework-kel lehetőség nyílik EMF példánymodellek definiálására a lekérdezések eredményéből, amiket utána akár módosíthatunk is.

### .eiqsnapshot fájl készítése

Ezeknek a kreált példánymodelleknek a kiterjesztése .eiqsnaphot, és a Query Explorer-ben hozhatjuk létre őket jobb gombbal a lekérdezés eredményén kattintva, majd a „Save EMF-IncQuery snapshot”-ra kattintva, ahogy az a következő képen is látszik:



25. ábra - EMF-IncQuery snapshot készítése

A snaphot fájl elkészítésekor két lehetőségünk van:

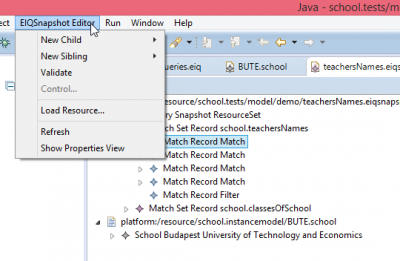
* Készítünk egy új .eiqsnapshot fájlt, amiben csak az aktuális lekérdezés eredménye szerepel példánymodellként
* Vagy egy már létező snapshot fájlhoz adjuk hozzá a lekérdezésünk eredményét

|  |  |
| --- | --- |
| Newsnapshot.png  26. ábra - Új snapshot fájl létrehozása | Existingsnapshot.png  27. ábra - Létező snapshot fájl használata |

### EIQ Snapshot Editor

Mint a neve is elárulja, ezzel az editorral a .eiqsnapshot fájlunkat szerkeszthetjük a megszokott EMF editor-os műveletekkel:

* Create new child
* Create new sibling
* Validate
* Load resource
* Refresh (model)
* Show properties view (még több információ, kontextusfüggő)
  + pattern qualified name
  + role
  + derived value
  + parameter name
  + value



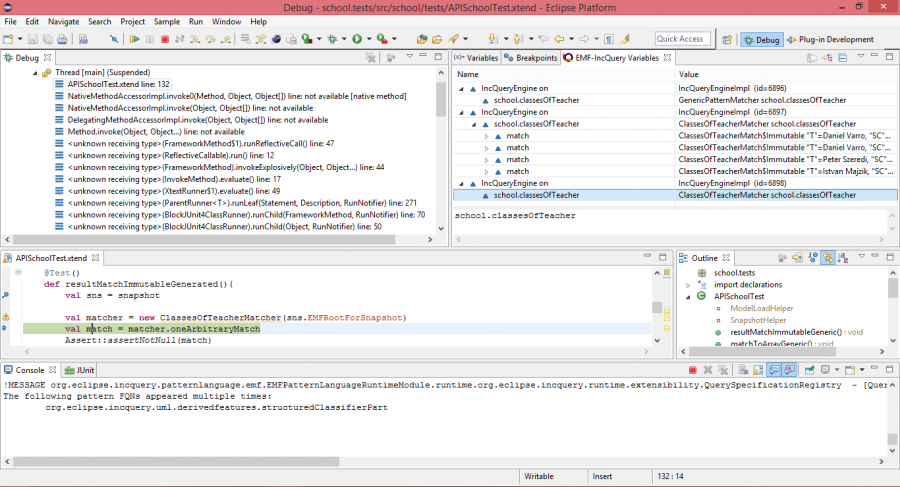
28. ábra - Az EIQSnapshot Editor

## Debugger Tooling

Az EMF-IncQuery Debugger Tooling, mint minden hasonló eszköz, a program működése során előforduló állapotok felfedezésére szolgál. A programkódban töréspontokat helyezhetünk el, majd vizsgálhatjuk a működést:

* példánymodell tartalmának megfigyelése
* EMF-IncQuery Pattern Matcher Engine belső állapotainak változása
* EMF-IncQuery Rule Engine állapotainak megfigyelése

Mivel ez a funkció nem elérhető az EMF-IncQuery-ben, csak a forráskódból fordítva, a továbbiakban nem kerül tesztelésre.



29. ábra - EMF-IncQuery Debugger Tooling

# Eclipse alapú eszközök grafikus felületének automatikus tesztelése – korábbi munkák áttekintése, összegzése

**Szabó Rudi (2011):**

Jnario használata SWTBot-tal, köztes réteg bevezetése a könnyebb használat érdekében. A tesztesetek leírására saját nyelv kitalálásának a gondolata, de nem érte volna meg, ezért a Gherkin nyelv használatát válaszotta. Az egyszerű nyelv miatt, a felhasználói dokumentáció írása is könnyebb. Ezekre az eszközökre épül a Jnario, mint tesztspecifikációt futtató keretrendszer és az XText segítségével JUnit kompatibilis Java kódot generál.



Tapasztalatok:

* SWTBot közvetlen használata esetén az asszertálási hiba okának megfogalmazása a fejlesztőn múlik, a hibás kódra viszont gyorsan lehet ugrani. Leírás nélkül nehezen azonosítható a hiba oka.
* Jnarioval a szöveges leírásból generálódik a tesztkód, így az mindig szinkronban van. A hiba helye is rögtön látható és viszonylag értelmes a hibaüzenet.
* Details/Filters táblázatának kezelése nincs implementálva
* Lekérdezés szerkesztő, paraméteres futások nem a diplomaterv céljai, ezeknek a tesztelése nem megoldott
* SWTBotRecorder közben jelent meg, egyszerűsíthetne a rendszeren
* Fedettség méréssel a diplomaterv nem foglalkozik

**Sas Csaba (2014):**

Jubula és Windows Tester Pro kipróbálása konklúzió nélkül

Fő tevékenység: A fentebb említett szakdolgozat folytatása, Jnairo használatával. Hiányzó elemek lekezelésének kiegészítése. Már az egyes QueryExplorer-es legördülő menük és combobox-ok is megjelennek a modellben.

Tapasztalatok:

* Itt már van kódfedettség vizsgálat. A megcélzott területen, vagyis a QueryExplorer és varázslók tekintetében magas fedettség elérése.



* A funkcionális tesztek az előbbi felsorolás lapján születtek. A tesztesetek a lekérdező nyelv szerkesztőjének felhasználói felületére nem terjednek ki (automatikus szöveg kiegészítés, validáció), továbbá a Rete Visualizer nézetre és az IncQuery Viewers Sandbox nézetre sem.
* SWBotRecorder kipróblása, de a grafikus felületen nem minden objektumot tud azonosítani, ezt célszerű továbbfejleszteni.

**Koska Sándor (2012):**

Különböző grafikus felhasználói felülett tesztelők összehasonlítása egy egyszerűbb példn keresztül, nem az EMF-IncQuery-re koncentrálva.

SWTBot:

* erős Java programozói tudás kell hozzá
* pozitív összkép
* natív funkciót nincsenek benne az API által lefedett GUI objektumtérben, mint például a „File Open/Save dialog” ablak kezelése.
* mindent programozással kell megoldani a fejlesztőnek, úgy, hogy akár nem is látja, és nem hajt végre semmilyen GUI beavatkozást a tesztelendő alkalmazáson

WindowTester Pro:

* tesztesetek implementálásához általában véve elégséges a Java programozási nyelvet minimális szinten ismerni, és az Eclipse környezet felhasználói ismerete szükséges hozzá
* a kiértékelés eredménye jól nyomon követhető, a hibás lefutások visszajelzése alapján könnyen megtalálható a hiba oka
* bonyolultabb tesztek futtatása már problémákba ütközhet, magas szintű Java programozási tudás kellhet hozzá, és a rendszer jobb ismerete
* szükséges kód nagy részének az automatikus legenerálásával, és ezt kézenfekvő módon biztosítja a tesztelendő alkalmazáson véghezvitt GUI események felismerésével

Jubula:

* egyáltalán nem igényel programozói tudást
* néhány bosszantó hiba, ami egyrészt megkerülhetetlen másrészt a fejlesztő energiájába kerül megkerülni őket

Elmondható, hogy mindegyik eszköz más esetben használható, már csak a különböző képzetség igénylés miatt is. Az SWTBot és WindowTester fontos előnye, hogy JUnit tesztet generálnak, ezek Continous Integration esetekben valószínűleg előnyösebbek.

|  |  |  |  |
| --- | --- | --- | --- |
| **Összefoglaló táblázat** | | | |
|  | **SWTBot** | **WindowTester** | **Jubula** |
| **Szoftver elérhetősége** | Eclipse Plug-in | Eclipse Plug-in | Eclipse Plug-in és Önálló telepítő csomag |
| **Tesztelendő alkalmazás állapota** | Csak a futtatáshoz kell a kész alkalmazás | Szükséges a kész alkalmazás a teszteset implementáláshoz | Modellváz készítés után kell csak a futtatható alkalmazás |
| **Szükséges előképzettség** | Haladó Java ismeret | Alap Java ismeret | Nem kell programozói képzettség |
| **GUI objektumok leképezéséhez nyújtott támogatás** | Beépített API | Capture után automata kódgenerálás + beépített API | Modellezéshez beépített modulok + Object mapping |
| **Eseményre várakozás kezelése** | Beépített metódusok, alapesetben van várakozás | Beépített metódusok, alapesetben van várakozás | Beépített modul, nincs automatikus várakozás |
| **Kiértékelési lehetőségek** | Többféle feltételvizsgálat, egyedi komplex vizsgálatok is | Többféle feltételvizsgálat, egyedi komplex vizsgálatok is | Többféle feltételvizsgálat |
| **Teszteset futtatás** | JUnit keretrendszerben, automatizálható, kezdő és lefutás utáni feladatok beállítására lehetőség, standard JUnit kiértékelés | JUnit keretrendszerben, automatizálható, kezdő környezet beállítási lehetőség, standard JUnit kiértékelés | Saját végrehajtó környezetében, külön ágensen futtatva az AUT-ot, fastruktúrában a futtatás eredménye |
| **Kezelési tapasztalatok** | Több hasonló teszteset után gyorsan lehet dolgozni benne, és kényelmes az API használata | A generált kód sokat gyorsít egy eset elkészítésén, az API jól kezelhető. | Jól kezelhető modellezés, de nem feltétlenül egyértelmű minden objektum megtalálása. |

# Az RCP Testing Tool

Az RCP Testing Tool egy automatikus grafikus felhasználói interfészt tesztelő eszköz Eclipse alapú alkalmazásokhoz.

**Fő képességek:**

* Teszteset kreálás lehetősége: Lehetőség van felhasználók tevékenységének rögzítésére egy programban, valamint a tesztelendő szoftver állapotainak elmentésére és annak további felhasználására.
* Eclipse technológiák támogatása: Mint például a Workspace figyelése a tesztelés során.
* Intelligens futás: UI események automatikus triggerelése.
* Értelmes eredmények: Az operációs rendszer működéséből adódó esetleges fals eredmények meg nem jelenítése. Vagyis csak releváns eredmények mutatása.
* Karbantarthatóság: A tesztesetek könnyű változtatása a UI változásaival együtt. Az RCPTT verziókövető rendszer barát is emellett.
* Bővíthetőség: A tool API-val ellátott például különböző jelentések készítésének támogatására.

**Workflow (Működés folyamata)**

Egy tipikus wokflow, amivel a grafikus interfész működését ellenőrizhetjük, így nézhet ki:

* Egy alkalmazás állapot rögzítése
* Teszt tevékenységek rögzítése (lényegében, hogy mikor hova kattintunk, és mit választunk ki)
* Asszertációk írása a kapott és elvárt eredmények összehasonlítására

# Tesztesetek continous integration környezetbe való áthelyzése

## qgears-es megoldás

## RCPTT Runner és Maven Plugin használata

## Konklúzió

# Továbbfejlesztési lehetőségek

# Köszönetnyilvánítás

A köszönetnyilvánítás nem kötelező, akár törölhető is. Ha a szerző szükségét érzi, itt lehet köszönetet nyilvánítani azoknak, akik hozzájárultak munkájukkal ahhoz, hogy a hallgató a szakdolgozatban vagy diplomamunkában leírt feladatokat sikeresen elvégezze. A konzulensnek való köszönetnyilvánítás sem kötelező, a konzulensnek hivatalosan is dolga, hogy a hallgatót konzultálja.

# Ábrák jegyzéke

[1.1. ábra. Példa képaláírásra 8](#_Toc396824939)

# Táblázatok jegyzéke

[1.1. táblázat. Példa táblázat feliratára 8](#_Toc396824940)

# Irodalomjegyzék

1. Jeney Gábor, Hogyan néz ki egy igényes dokumentum? Néhány szóban az alapvető tipográﬁai szabályokról, Budapesti Műszaki és Gazdaságtudományi Egyetem, Híradástechnikai Tanszék, Budapest, 2007. május 9., online: <http://mcl.hu/~jeneyg/foliak.pdf>
2. William Strunk Jr., E. B. White, The Elements of Style, Fourth Edition, Longman, 4th edition, 1999.
3. Levendovszky, J., Jereb, L., Elek, Zs., Vesztergombi, Gy., Adaptive statistical algorithms in network reliability analysis, Performance Evaluation – Elsevier, Vol. 48, 2002, pp. 225-236
4. National Instruments, LabVIEW grafikus fejlesztői környezet leírása, <http://www.ni.com/> (2014. aug.)
5. EMF-IncQuery – FeautreSet and Testing - Query Explorer <http://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/QueryExplorer>
6. EMF-IncQuery – FeatureSet and Testing - Data binding <https://wiki.eclipse.org/EMFIncQuery/UserDocumentation/Databinding>
7. EMF-IncQuery – FeatureSet and Testing - Preferences <https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/Preferences>
8. EMF-IncQuery – FeatureSet and Testing – Validation framework <https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/Validationframework>
9. EMF-IncQuery – FeatureSet and Testing – Viewers <https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/Viewers>
10. EMF-IncQuery – FeatureSet and Testing – Rete Visualizer https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/Rete
11. EMF-IncQuery – FeatureSet and Testing – Testing framework https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/Testingframework
12. EMF-IncQuery – FeatureSet and Testing – IncQuery Debugger View https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/Features/Debug
13. EMF-IncQuery – FeatureSet and Testing – Xcore integration https://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/Features/Xcore

# Függelék

A függelék szövege.

1. Continous Integration, azaz folytonos integráció megvalósított-e valamilyen buildszerveren. [↑](#footnote-ref-1)
2. Continous Integration, azaz folytonos integráció megvalósított-e valamilyen buildszerveren. [↑](#footnote-ref-2)