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

Konzulensek

Dr. Hegedüs Ábel

2015

Tartalomjegyzék

[Összefoglaló 9](#_Toc419141389)

[Abstract 10](#_Toc419141390)

[1. Bevezetés 11](#_Toc419141391)

[1.1. Formázási tudnivalók 11](#_Toc419141392)

[1.1.1. Címsorok 11](#_Toc419141393)

[1.1.2. Képek 11](#_Toc419141394)

[1.1.3. Táblázatok 11](#_Toc419141395)

[1.1.4. Kódrészletek 12](#_Toc419141396)

[1.1.5. Irodalomjegyzék 12](#_Toc419141397)

[1.1.6. Margók 13](#_Toc419141398)

[2. Utolsó simítások 14](#_Toc419141399)

[3. Összefoglalás 15](#_Toc419141400)

[4. Bevezetés 16](#_Toc419141401)

[4.1. A feladat célja 16](#_Toc419141402)

[4.2. A szakdolgozat felépítése 16](#_Toc419141403)

[5. EMF-IncQuery, a gráflekérdező keretrendszer 18](#_Toc419141404)

[6. Eclipse alapú eszközök grafikus felületének automatikus tesztelése 19](#_Toc419141405)

[6.1. Irodalomkutatás az eddig használt tesztelő alkalmazásokról 19](#_Toc419141406)

[6.1.1. Jnairo használata SWTBot-tal 19](#_Toc419141407)

[6.1.2. SWTBot, WindowTester Pro és Jubula összehasonlítása 21](#_Toc419141408)

[6.2. Az RCP Testing Tool 23](#_Toc419141409)

[6.2.1. Az RCP Testing Tool fő képességei 23](#_Toc419141410)

[6.2.2. Egy egyszerű teszteset készítése az RCPTT-vel 23](#_Toc419141411)

[7. Az EMF-IncQuery tesztkészletének felderítése és a további lépések meghatározása 26](#_Toc419141412)

[7.1. A tesztelésről röviden 26](#_Toc419141413)

[7.2. A tesztkészlet felderítése 27](#_Toc419141414)

[7.3. A szakdolgozat további lépéseinek meghatározása 32](#_Toc419141415)

[8. Az EMF-IncQuery GUI komponensei 34](#_Toc419141416)

[8.1. Query Editor, a lekérdezés szerkesztő 34](#_Toc419141417)

[8.2. Query Explorer, a lekérdezés menedzser 34](#_Toc419141418)

[8.2.1. EMF-IncQuery-t tartalmazó kontextus függő menü 35](#_Toc419141419)

[8.2.2. „Magic Green Button”, a modell- és mintabetöltő 36](#_Toc419141420)

[8.2.3. Pattern registry, a mintamenedzser 37](#_Toc419141421)

[8.2.4. Result viewer 38](#_Toc419141422)

[8.2.5. Details/Filters 39](#_Toc419141423)

[8.2.6. View menu 40](#_Toc419141424)

[8.3. Preferences, az EMF-IncQuery beállítási lehetőségei 41](#_Toc419141425)

[8.4. Validation framework, az EMF-IncQuery validációs megoldása 43](#_Toc419141426)

[8.5. Viewers 45](#_Toc419141427)

[8.5.1. Viewers Sandbox 45](#_Toc419141428)

[8.6. Rete Visualizer 47](#_Toc419141429)

[8.7. Testing framework 48](#_Toc419141430)

[8.7.1. .eiqsnapshot fájl készítése 49](#_Toc419141431)

[8.7.2. EIQ Snapshot Editor 50](#_Toc419141432)

[8.8. Debugger Tooling 51](#_Toc419141433)

[9. Tesztesetek felvétele az RCP Testing Tool segítségével 52](#_Toc419141434)

[9.1. A Workbench felvétele 52](#_Toc419141435)

[9.2. Különböző Workspace-ek felvétele 53](#_Toc419141436)

[9.3. A kezdeti állapotot beállító szkriptek felvétele 54](#_Toc419141437)

[9.4. Az elkészült tesztesetek 54](#_Toc419141438)

[10. Tesztesetek integrálása folytonos integrációs rendszerhez 56](#_Toc419141439)

[10.1. Az Apache Maven és a Jenkins 56](#_Toc419141440)

[10.2. Külső megoldás használata a tesztek integrálásához 57](#_Toc419141441)

[10.3. Az RCPTT Runner használata a tesztek integrálásához 58](#_Toc419141442)

[10.3.1. Az RCPTT Runner bemutatása 58](#_Toc419141443)

[10.3.2. A Jenkins konfigurálása 60](#_Toc419141444)

[10.4. Konklúzió 61](#_Toc419141445)

[11. Összefoglalás és továbbfejlesztési lehetőségek 63](#_Toc419141446)

[Köszönetnyilvánítás 64](#_Toc419141447)

[Ábrák jegyzéke 65](#_Toc419141448)

[Táblázatok jegyzéke 67](#_Toc419141449)

[Irodalomjegyzék 68](#_Toc419141450)

[Függelék 70](#_Toc419141451)

Hallgatói nyilatkozat

Alulírott , szigorló hallgató kijelentem, hogy ezt a szakdolgozatot 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. 05. 11.

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. táblázat mintájára érdemes elkészíteni.

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

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

## A feladat célja

Az Eclipse Platform és az erre épülő alkalmazások meghatározó szerepet foglalnak el a szoftverfejlesztés iparágban. A nyílt-forráskódú szoftverek nagy része is erre a platformra épül. Ezek közül kiemelkedő fontosságúak, és manapság egyre elterjedtebbek az úgynevezett Eclipse Modeling Platform berkein belül elkészülő programok, amelyek valamilyen modellvezérelt szoftverfejlesztési paradigmára épülnek.

Ezek a szoftverek mind használatosak a minőségi és biztonsági szempontból kritikus beágyazott rendszereknél, valamint a komplex, szolgáltatásorientált szoftverrendszerek területén is. Szakdolgozatom témája is egy ilyen alkalmazás, az EMF-IncQuery tesztelése.

A szakdolgozat keretein belül megvalósítandó feladatok:

* Irodalomkutatás végzése a Java alapú szoftverek, illetve eclipse beépülő modulok tesztelésére alkalmas technológiák között.
* Az EMF-IncQuery-hez készült eddigi tesztkészlet felmérése, tesztkatalógus készítése.
* Javaslattétel a tesztkészlet bővítésére az első pontban felsorolt technológiák valamelyikének használatával.
* Tesztkészlet integrálása folytonos integrációs környezetbe.
* Munka értékelése, kód- és funkciófedettség vizsgálata, továbbfejlesztési lehetőségek javaslása.

## A szakdolgozat felépítése

Ebben a fejezetben bemutatom a szakdolgozatom felépítését. Igyekeztem jól elkülöníteni a munkám irodalomkutatási és tényleges fejlesztési részét. Ennek megfelelően a szakdolgozatom felépítse a következő.

A dokumentum első felében bemutatom a tesztelt eszközt, az EMF-IncQuery-t, és ismertetem alapvető funkcióit. Ezen kívül irodalomkutatást végzek a grafikus felhasználói interfészt tesztelő eszközök körében. Ez kiterjed egyrészt korábbi szakdolgozatok átnézésében, az eddigi tapasztalatok összegzésében, valamint egy újonnan megjelent eszköz, az RCP Testing Tool kipróbálásában.

Mivel a feladatom a keretrendszer integrációs tesztelése, szükségszerű volt az tesztelés alapfogalmaival megismerkedni. Erről is írtam egy rövid összefoglalót *A tesztelésről röviden* fejezetben.

A szakdolgozat második felében felmérem az EMF-IncQuery tesztkészletét.

Use-case-ek gyűjtése. Tesztkészlet elkészítése RCPTT-vel.

Tesztkészlet hozzáadása folytonos integrációs rendszerhez. A qgears-es megoldás bemutatása. Az RCPTT Runner bemutatása.

# 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.

Az Eclipse Modeling Platform fontos része az EMF, vagyis az Eclipse Modeling Framework. Ennek a keretrendszernek a segítségével lehetőségünk van metamodellek definiálására, ezekből pedig példánymodellek származtatására. Az EMF támogatja az elkészített modellekből Java kód generálását is sok más egyéb mellett.

Az EMF-IncQuery-vel ilyen EMF modellek felett fogalmazhatunk meg lekérdezéseket, 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)
* EMF-IncQuery integrálása más alkalmazásokba a különböző API-k használatával

Az EMF-IncQuery, a lekérdezőnyelv különböző annotációinak köszönhetően képes az eredményeket többféleképpen is megjeleníteni. Az alapértelmezett megjelenítés a fa struktúrát használja, ahol lehetőség van különböző feltételek mellett a találati lista szűrésére vagy a részletek megjelenítésére. Ezen kívül különböző gráf nézetek is elérhetőek, ami speciális – például UML – modellek megjelenítése esetén jöhet jól.

# Eclipse alapú eszközök grafikus felületének automatikus tesztelése

Az Eclipse egy nyílt forráskodó, platformfüggetlen keretrendszer, ami főként Java nyelven íródott. Egy Eclipse alkalmazás tulajdonképpen különböző View-k, vagyis nézetek összessége. Ezeknek felhasználói felülete az *SWT*-n (Standard Widget Toolkit) és *JFace*-n alapul. Utóbbinak köszönhetően ezek a nézetek ugyanarra a sémára épülnek és így nagyon hasonlóak. Ennek köszönhető, hogy számos alkalmazás elérhető, amelyekkel ilyen felületeket lehet automatizáltan tesztelni.

## Irodalomkutatás az eddig használt tesztelő alkalmazásokról

Korábban többen is foglalkoztak szakdolgozat témaként automatikus felhasználói felület tesztelő eszközökkel, a következő alfejezetekben ezek összefoglalása következik.

### Jnairo használata SWTBot-tal

Az első törekvés az EMF-IncQuery felhasználói felületének tesztelésére a *Jnario* és *SWTBot* együttes használatával történt. A tesztkörnyezet ekkor még saját kézzel kellett összerakni különböző komponensekből, nem volt kiforrott technológia. A tesztesetek leírására a Gherkin nyelvet használták. Ez egy egyszerű nyelv, aminek köszönhetően 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. Az alábbi képen látható, hogy hogyan épült fel a tesztkörnyezet. [6]



2. ábra - A Jnario-s tesztkeretrendszer felépítése

Kezdetben a következő problémák adódtak ezzel a rendszerrel:

* *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.
* A szűrő/részletező funkció táblázatának kezelése nincs implementálva.
* Lekérdezés szerkesztő, paraméteres futások tesztelése nem megoldott.

Ezután, egy következő szakdolgozat keretében ezeknek a hibáknak, hiányosságoknak nagy része megoldásra került, és a tesztkészlet is ki lett bővítve.[7]

A dolgozat során lefedett funkciók:

* EMF-IncQuery Project létrehozása
* EMF-IncQuery Generator Model létrehozása
* EMF-IncQuery Query Definition létrehozása
* EMF Ecore model létrehozása
* EMF Generator model létrehozása
* Empty EMF Project létrehozása
* Example EMF Model Creation Wizard felület működése
* Minták futtatása, kiértékelése a Query Explorer segítségével

A tesztesetek a lekérdező nyelv szerkesztőjének felhasználói felületére nem terjednek ki (automatikus szöveg kiegészítés, validáció), továbbá a különböző gráf megjelenítők tesztelésére sem.

Ennek ellenére az EMF-IncQuery egyes komponenseinél magas kódfedettség lett elérve, ahogy az a képen is látszik, de ez minden esetre nem érinti az összes egységet.



3. ábra - Kódfedettség *Jnario* és *SWTBot* használatával

A tesztkörnyezet legnagyobb problémája, hogy a tesztek megírása körülményes a felhasználói interakciók rögzítése miatt. Az SWTBot-hoz elérhető az úgynevezett *SWTBotRecorder*, ami felhasználói felületen való kattintásokat és billentyűzetlenyomásokat rögzítené, de nem minden objektumot ismer fel és így alapesetben használhatatlan sok komponens tesztelésére.

### SWTBot, WindowTester Pro és Jubula összehasonlítása

Egy másik tanszéki szakdolgozat is foglalkozott különböző grafikus felhasználói felület tesztelő programok összehasonlításával, de nem az EMF-IncQuery-re koncentrálva, hanem csak egy egyszerű tesztkészletet felállítva, és azon próbálgatva az egyes eszközöket. A dolgozat alapján összegyűjtöttem az egyes eszközökről a tapasztalatokat, és levontam a konklúziót. A fejezet végre beszúrtam egy, a szakdolgozatból vett összehasonlító táblázatot a gyorsabb áttekintés érdekében. [8]

SWTBot:

* erős Java programozói tudást igényel
* natív funkciót nem elérhetők használható objektumként, trükközni kell használatukért
* 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 szintű 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
* teszteléshez szükséges kód nagy részének automatikus legenerálása 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

|  |  |  |  |
| --- | --- | --- | --- |
| **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. |

2. táblázat - SWTBot, WindowTester Pro és Jubula összehasonlító táblázat[8]

Elmondható, hogy mindegyik eszköz más esetben használható, már csak a tesztelő személy programozói képzettsége miatt is. Az SWTBot és WindowTester Pro fontos előnye, hogy JUnit tesztet generálnak, így ezek folytonos integrációs környezetben valószínűleg hasznosabbak lehetnek. Ugyanakkor mindegyik eszköznek van valami hátránya, ami miatt érdemes más tesztelő eszköz után kutatni.

## Az RCP Testing Tool

Az RCP Testing Tool (röviden RCPTT) egy automatikus, grafikus felhasználói interfészt tesztelő eszköz, eclipse alapú alkalmazásokhoz. Ebben a fejezetben összegyűjtöm a program fő funkcióit és egy egyszerű példán keresztül bemutatom, hogy hogyan készíthetünk teszteket ezzel az alkalmazással.

### Az RCP Testing Tool fő képességei

Az RCPTT számos, a tesztelést segítő funkcióval rendelkezik:

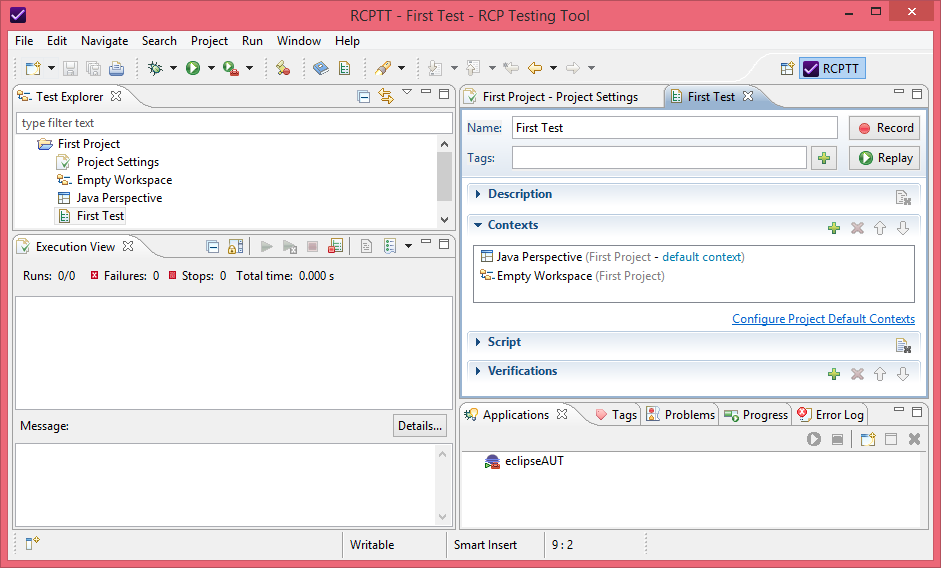
* Teszteset készítés lehetősége manuálisan vagy a felhasználói interakciók automatikus rögzítésével
* Az Eclipse Platform fogalmainak támogatása, mint például a workspace, workbench, preferences, stb. Ezen kívül képes a GEF/GMF/Graphiti diagramokat is kezelni, ami nagyon hasznos, ugyanis az EMF-IncQuery is használ ilyeneket.
* Az intelligens futtatásnak köszönhetően az aszinkron vagy háttérműveletekre való várakozás is megoldott. Például a tesztelt alkalmazásnál megvárja a workspace build-elését és csak utána kezdi a következő műveletet.
* Operációs rendszer és képernyőméret független tesztelés és teszteredmények mutatása. Ez nagyon hasznos abban az esetben, ha folytonos integrációs környezetben akarjuk használni az RCPTT-t. Ugyanis a build szervereken általában Linux operációs rendszer fut, a fejlesztés pedig gyakran Windows operációs rendszeren történik. Így a Windows-on írt tesztjeinket gond nélkül futtathatjuk Linuxon is.
* Az RCPTT-vel írt tesztek könnyen karbantarthatóak és könnyen módosíthatóak a tesztelt rendszer változása esetén. A tesztelés során létrejövő fájlok egyszerű felépítésűek, így a verziókövetés is könnyedén megoldható.
* Rendelkezik folytonos integrációs támogatással, így az elkészült tesztek könnyen integrálhatók egy szoftverfejlesztési rendszer részeként.

### Egy egyszerű teszteset készítése az RCPTT-vel

Az RCPTT is egy eclipse alkalmazás, tehát a programot elindítva nagyon ismerős felület fogad minket. A tesztelést, mint mindig, egy Projekt létrehozásával kezdjük, a példámban ez a First Project nevet kapta. Mielőtt nekikezdünk a teszteset elkészítésének, érdemes és általában szükséges is különféle futtatási kontextusokat megadni. Ez annyit jelent, hogy az RCPTT a megadott környezet előállítja a tesztelt alkalmazásban. Ilyen egyszerű kontextus lehet például a perspektíva beállítása, ami lényegében nézetek összessége. Tehát megadatjuk, hogy milyen nézetek legyenek elérhetőek a teszt futtatásakor. Ezen kívül a teszthez használt workspace-t is beállíthatjuk. Természetesen az eclipse alkalmazást is meg kell adnunk, amit használni fogunk. A példámban ez az eclipseAUT nevet kapta.

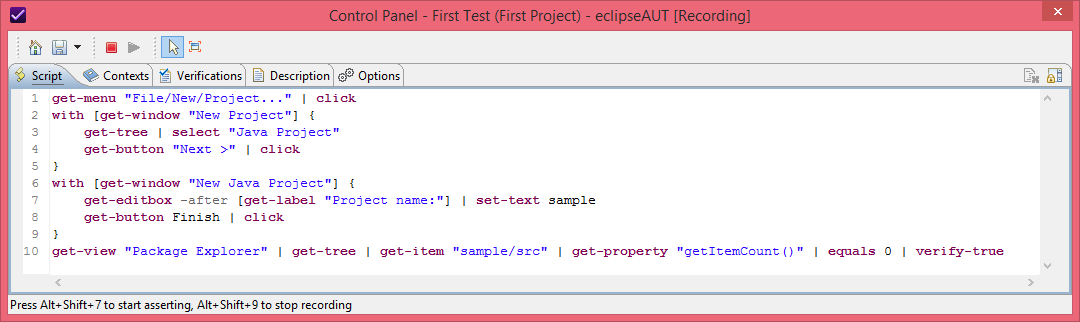
A bemutató példámban egy olyan tesztet készítek, ami létrehoz egy Java projektet egy üres workspace-ben, és azt vizsgálja, hogy az adott projekten belül létrejött-e egy üres src mappa.

Ehhez futtatási környezetként Java perspektívát és egy üres workspace-t definiálok, majd létrehozom a tesztesetet First Test néven. Ezek a lépések jól látszanak az alábbi képen.



4. ábra - Az RCP Testing Tool kinézete

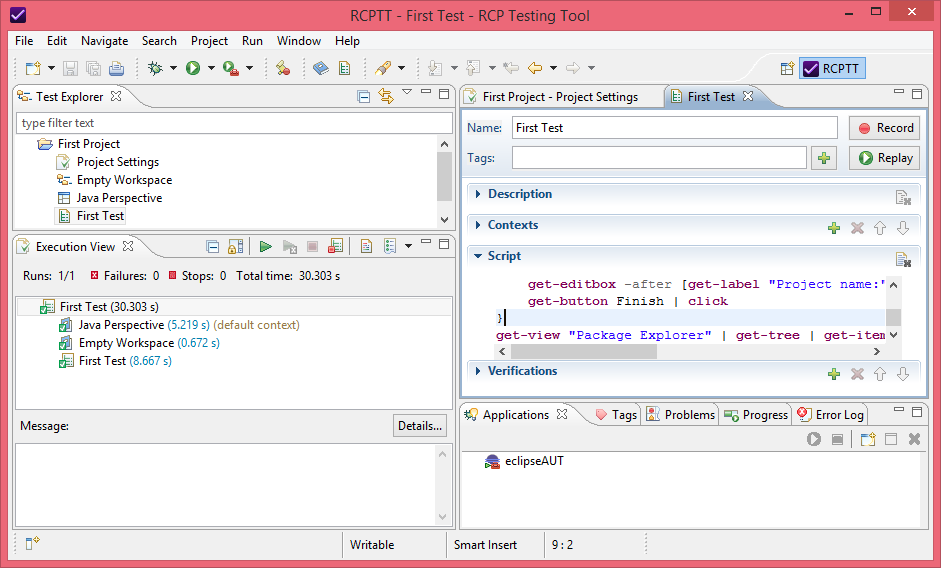
A képen látható Record gombra kattintva vehetjük fel a tesztesetet. Ekkor elindul a tesztelendő alkalmazás, jelen esetben az eclipseAUT és az úgynevezett Control Panel, amiben rögzítődnek a felhasználói interakciók.



5. ábra - Az RCP Testing Tool Control Panel

A képen jól látszik, hogy ez a főként számítógép számára hasznos szkript nyelv emberi szemmel is jól olvasható és akár módosítható is utólag. A  gombra kattintva válthatunk asszertálási módba, vagyis itt ellenőrizhetjük, hogy a felhasználói interakciók eredménye helyes-e. A képen látható utolsó sor kód ellenőrzi a fentebb említett tesztelési feltételt jelen esetben.

Ezután már egyszerűen futtathatjuk elkészült tesztünket, és esetleges hiba esetén is informatív üzenetet kapunk.



6. ábra - Tesztek futtatása az RCPTT-vel

# Az EMF-IncQuery tesztkészletének felderítése és a további lépések meghatározása

Ebben a fejezetben röviden összefoglalom a munkámhoz szükséges tesztelési tudnivalókat, majd eszerint bemutatom, kategorizálom az EMF-IncQuery-hez eddig elkészült teszteket. A tesztkészlet felmérésével minden információ rendelkezésre áll a további teendők meghatározásához, így a fejezet végén ezek összegyűjtése található.

## 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: [9]

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

A komponensteszt csak a rendszer egy komponensét teszteli önmagában. Egy ilyen komponens lehet tipikusan valamilyen vizuális elem megjelenítésért felelős. Az integrációs teszt kettő vagy több komponens együttműködését teszteli. A rendszerteszt az egész rendszert, tehát minden komponenst együtt tesztel. Ez első három tesztet fejlesztői teszteknek nevezzük, mert ezeket a fejlesztő cég készíti, az átvételi teszt során a pedig a végfelhasználók már a kész rendszert tesztelik. Ezek általában időrendben is így követik egymást.

A komponensteszten belül – attól függően, hogy mit tesztel – a következő két fajta tesztet különböztetjük meg:

* unit-teszt,
* modulteszt.

A unit-teszttel metódusokat tesztelünk általában. Azt kell figyelnünk, hogy egy adott függvény meghívására, az a megfelelő értékkel tér e vissza. A fejlesztő általában egyszerre készíti a komponenst és a hozzá tartozó tesztet, így rögtön láthatja ha hibát követett el. Ebben az esetben a tesztelő célja, hogy minél magasabb kód-fedettséget érjen el és lehetőleg minden egyes elágazást lefedjen tesztjeivel. Ezt fehér dobozos tesztelésnek nevezzük, mivel ismerjük a komponens, rendszer belső működését és ez alapján tesztelünk.

A modulteszttel általában a komponensünk teljesítménybeli képességeit szoktuk vizsgálni. Például, hogy van-e memóriaszivárgás vagy szűk keresztmetszet a rendszerben.

Az integrációs teszt során a komponensek közti függvényhívásokat, az operációs rendszer és a rendszer közti interfészt, illetve más rendszerek felé nyújtott interfészeket teszteljük. Az integrációs két legfontosabb fajtája:

* Komponens integrációs teszt: A komponensek közötti kommunikáció tesztje.
* Rendszer integrációs teszt: A rendszer és más rendszerek közötti kommunikáció tesztje.

Az integrációs teszt a rendszer egyes komponenseinek összeillesztése során keletkező hibákat keresi. Mivel a különböző programrészeket általában más-más programozók, vagy esetleg csapatok fejlesztették, ezért a kommunikáció hibák miatt súlyos problémák jöhetnek létre.

A fentebb említett fehér dobozos tesztelés ellentéte a fekete dobozos tesztelés. Ebben az esetben a tesztelő nem ismeri a komponensek, függvények belső működését, csak a tesztelt program funkcióit ismeri. Ebben az esetben azt ellenőrizzük, hogy adott események sorozatára az elvárt eredmények következnek-e be. Jól látszik, hogy a grafikus felhasználói felület tesztelése is ilyen tesztelés.

## A tesztkészlet felderítése

Az EMF-IncQuery-hez számos teszt készült az évek során, de ezek szétszórva találhatóak meg az interneten, különböző tárhelyeken, és különböző példány modelleket használnak a teszteléshez. Természetesen a tesztelés szintjében is különböznek, találhatunk mind komponens szintű unit teszteket, mind integrációs teszteket. 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
* Folytonos integrációs rendszerben 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.

Az EMF-IncQuery-hez tartozó tesztesetek három különböző repository-ban, vagyis tárhelyen találhatóak. Ezek a következők:

* incquery-core-repo: Ez egy nem hivatalos elnevezése a fő IncQuery repository-nak, itt található a fejlesztett forráskód is. Az itteni tesztek mind up-to-date-ek, ugyanis a rendszeren való változtatás után a tesztek futtatásához rögtön módosítani kell azokon, hogy sikeresen fussanak.
* eiq-examples-repo: Ezen a tárhelyen főként az évek alatt összegyűlt, IncQuery-hez készült példaalkalmazások találhatók. Ezek közül némelyik tartalmaz JUnit teszteket is. A régebbi tesztek közül vannak, amelyek nem működnek már az IncQuery újabb verziójával, így jelen esetben használhatatlanok.
* incquery-examples-repo: Ez egy viszonylag újonnan létrehozott tárhely, az eiq-examples-repo-hoz hasonló, de az itteni tesztek nagy része működik és az újonnan az eszközbe került funkciókat tesztelik, mint például az UML támogatás.

A tesztesetek között megkülönböztethetünk unit- és integrációs teszteket. A unit tesztek az EMF-IncQuery valamilyen komponensét tesztelik. Az incquery-core-repo-ban található a legtöbb ilyen teszt. Ezek a fejlesztés során készültek és például a mintaillesztési nyelvet vagy gráf bejárási algoritmusokat tesztelik. Ezek a tesztek tipikusan egy komponensre korlátozódnak.

A komponens tesztek közül még kiemelhető a network nevű modulteszt, ami a fejlesztett eszköz teljesítményét hivatott mérni. Ehhez 3 különböző méretű példánymodellt használ, és a kimenetén értesít a futási időről, valamint a felhasznált memóriáról. A teszt jellegéből adódóan, főként a mintaillesztési valamint lekérdező algoritmusokat tartalmazó komponenseket teszteli.

A komponens integrációs tesztek közül a bpm és school tesztek a legjelentősebbek. Ezek egy konkrét példán keresztül az egész rendszert tesztelik. A bpm egy üzleti modellt használ a lekérdezésekhez, a school pedig egy iskola példánymodellét. Ezek a tesztek a folytonos integrációs környezetben is használva vannak és a jelenlegi kód-fedettség nagy részét adják.

Az eiq-examples-repo, révén, hogy egy több éves tárhely, tartalmaz nagyon régi teszteket is. Ezek közül a derivedTest és a bpmn.tests teszt már elavult, az aktuális 1.0.0-ás EMF-IncQuery-vel való használathoz, erről egyértelmű hibaüzenet is kaptam futtatáskor. A derivedTest jelezte, hogy 0.6-os IncQuery-vel működik csak, a bpmn.tests pedig az importált metamodell feldolgozásával ütközött problémába.

Az alábbi táblázatokban megtekinthetők az összegyűjtött információk az egyes tesztekről strukturált formában.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 |
| ecorequery.tests | ecore.ecore | komponens integrációs teszt | Ecore.ecore modellen végzett lekérdezések | runtime (40%),  patternlanguage (40%) | eiq-examples-repo | Igen |

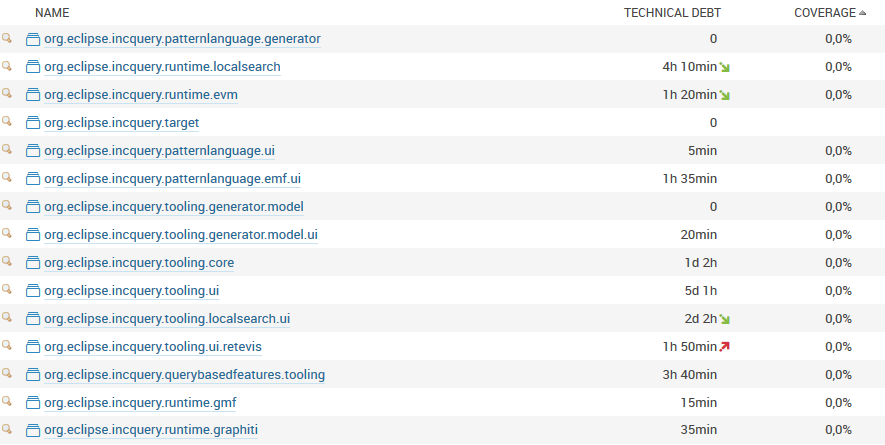
3. táblázat - EMF-IncQuery tesztkatalógus 1. rész

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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)** |
| 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%) | incquery-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%) | incquery-examples-repo | Igen |

4. táblázat - EMF-IncQuery tesztkatalógus 2. rész

## A szakdolgozat további lépéseinek meghatározása

A tesztkészlet felderítése során világossá vált, hogy a felhasználói felület megjelenítéséért felelős komponensek egyáltalán nincsenek tesztelve. Ez jól látszik a build.inf.mit.bme.hu-ról[[3]](#footnote-3) vett Sonar[[4]](#footnote-4) kód-fedettségi adatokból is. Az alábbi képen látható, hogy a „.ui”-ra végződő komponensek fedettsége mind 0%. Ezen kívül még több programtész is van, ami nincs fedve, de a kép csak szemléltető szándékkal van itt.



7. ábra - Részlet a Sonar kód-fedettségi adatokból

Ezután jogosan merül fel ötletként és elvárásként a grafikus felhasználói felület tesztelése. Ehhez, a már korábban bemutatott RCP Testing Tool-t fogom használni. Ahhoz, hogy ez kellően átgondoltan és minden komponensre kiterjedően történhessen, nem elég ad-hoc módon tesztelni a felületet. Szükséges a felhasználói felületen végezhető interakciók összegyűjtése és rendszerezése.

Miután megvannak a use-case-ek már csak az RCPTT-vel kel felvenni az egyes teszteseteket és kitalálni, hogy mire érdemes asszertálni, vagyis mi az elvárt eredmény az egyes interakciók után.

Önmagukban viszont az egyes tesztek nem sokat érnek, ha nem lehet hozzáadni őket egy folytonos integrációs rendszerhez, ahol a tesztelt eszköz legfrissebb verziójával futhatnak. Szóval a szakdolgozatom utolsó lépése ennek a megoldása volt. Mivel az RCPTT azon része, ami a tesztek build szerveren való automatikus futtatásáért felel, csak a szakdolgozat írása közben vált nyílt forráskódúvá, kezdetben egy másik megoldással próbálkoztam.

A szakdolgozatom további részében, miután összegyűjtöttem az EMF-IncQuery use-case-eit és az ezekhez tartozó teszteket megvalósítottam az RCPTT-vel, ezzel a két módszerrel oldom meg a tesztesetek integrálását, majd konklúziót vonok, hogy mik az egyes módszerek előnyei, illetve hátrányai. Illetve elemzem a kapott kód-fedettségi adatokat, hogy mennyit segítenek az egész eszköz tesztelésében az én saját integrációs tesztjeim.

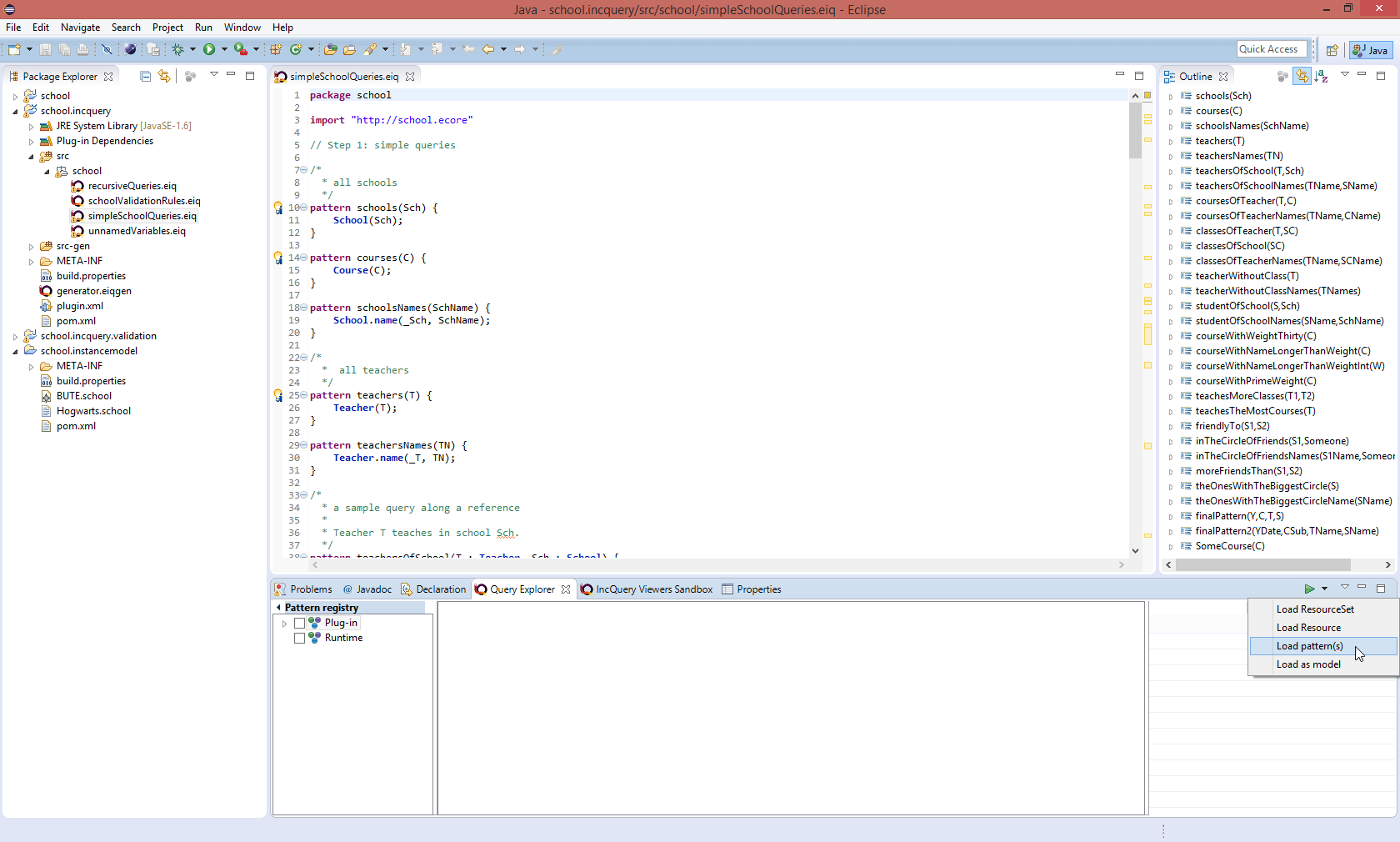
# Az EMF-IncQuery GUI komponensei

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 a felhasználói felület tesztelésének igénye. Ehhez először is össze kellett szedni az EMF-IncQuery grafikus felhasználói felületének elemeit, és az ehhez tartozó use-case-eket. Ez lényegében a lehetséges felhasználói interakciókat jelentette. A következőkben ezeknek a felsorolása következik. Az egyes fő fejezetek jelölik a GUI komponenst, alfejezetekként pedig az adott komponenssel végezhető műveletek találhatóak. [5]

## Query Editor, a lekérdezés szerkesztő

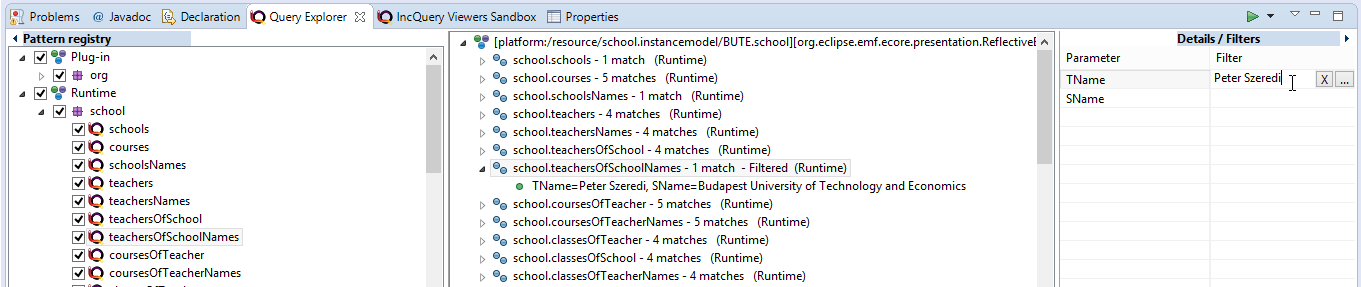
A Query Editor egy Xtext alapú szerkesztő, amivel EMF-IncQuery lekérdezéseket szerkeszthetünk. Ez a szerkesztő rengeteg fejlesztést segítő megoldást tartalmaz, mint például az automatikus kód kiegészítés, amit már más fejlesztő eszközökből is megismerhettünk. Mivel ennek a szerkesztőnek a vizsgálata 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.



8. ábra - A Query Editor kinézete

## Query Explorer, a lekérdezés menedzser

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. Az utóbbiak különböző grafikus szerkesztést tesznek lehetővé. A Query Explorer ezen kívül képes a lekérdezések utólagos feldolgozására is, például a találati lista szűrésére vagy a részletek megjelenítésére.

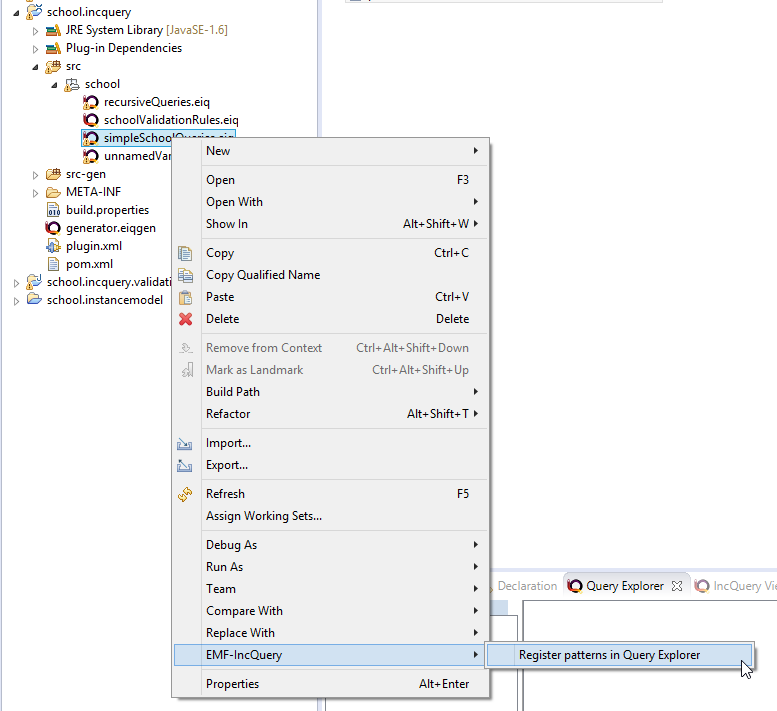


9. ábra - A Query Explorer kinézete

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

### EMF-IncQuery-t tartalmazó kontextus függő menü

Egy mintákat tartalmazó .eiq fájlra jobb gombbal kattintva a kontextus függő menüben a szokványos menüelemeken kívül - 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. Segítségével kiválaszthatjuk, hogy mely lekérdezések eredményét szeretnénk látni éppen.

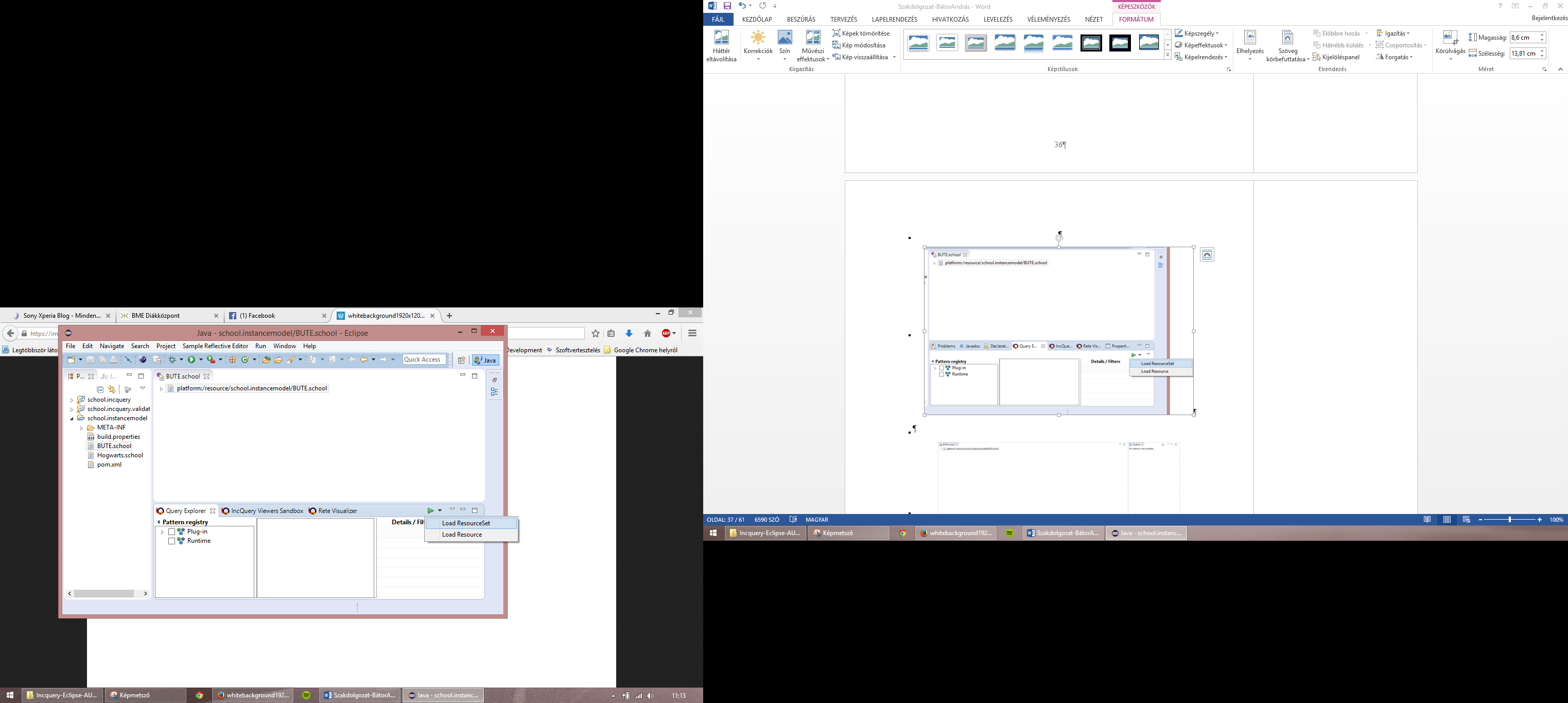


10. ábra - Pattern regisztrálása a Query Explorer-ben

### „Magic Green Button”, a modell- és mintabetöltő

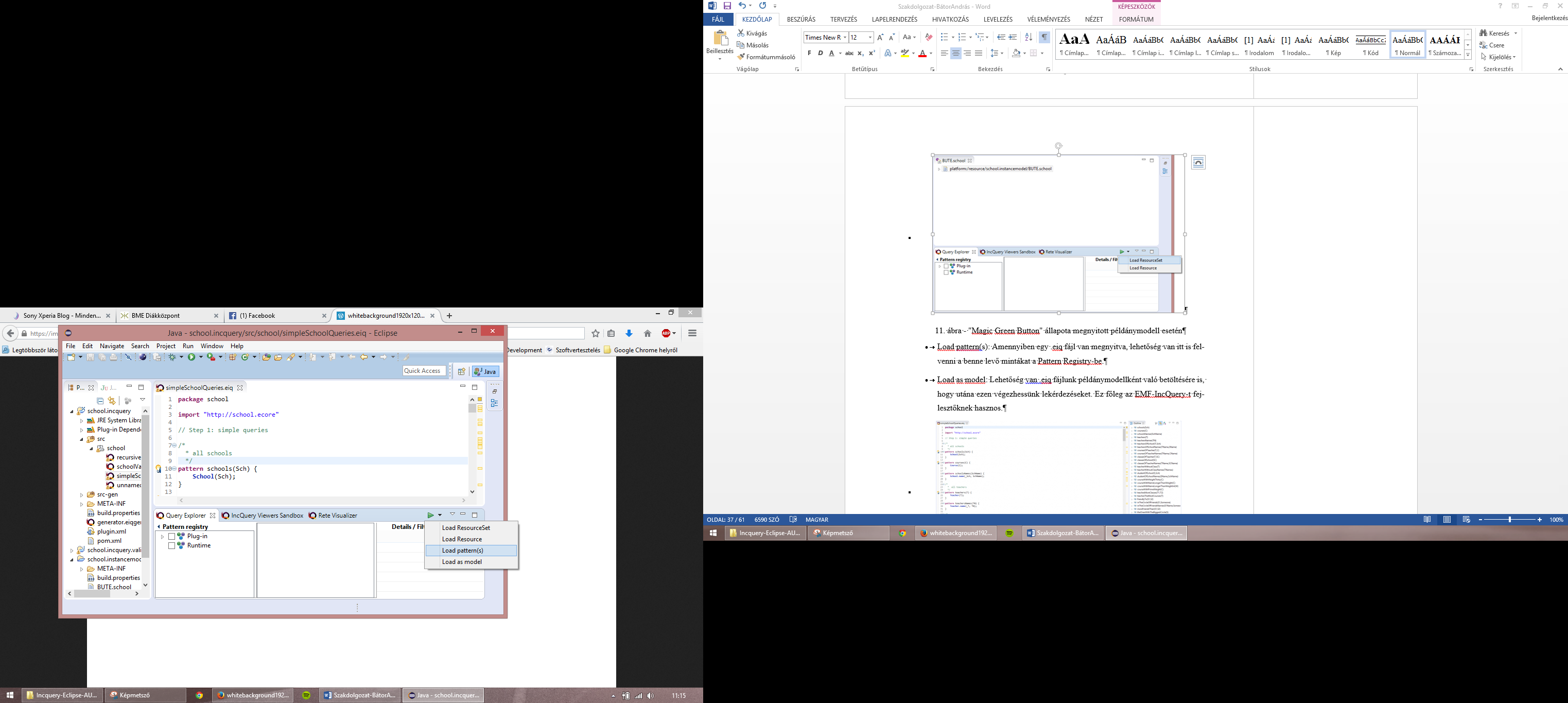
A Magic Green Button működése szintén kontextusfüggő. Miután a példánymodellünket megnyitottuk a Sample Reflective Ecore Model Editor-al a Query Explorer zöld gombja aktívvá válik és használatra kész. Szöveges szerkesztő esetén ez nem történik meg. A workbench pillanatnyi állásától függően ezután a következőkre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



11. 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.

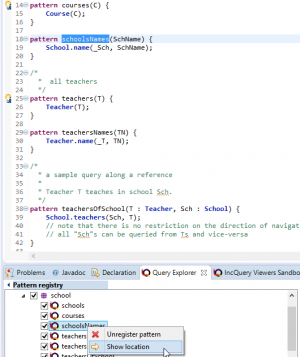


12. ábra - Magic Green Button állapota megnyitott .eiq fájl esetén

### Pattern registry, a mintamenedzser

A Query Explorer bal oldalán elhelyezkedő Pattern registry a betöltött mintákat mutatja, a View menu-ben beállított package hierarchiának megfelelően (lásd 8.2.6-os fejezet). Főbb funkciói 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 ugyanis ezek kellenek az eszköz műlödéséhez.
* Show location: pattern definiálásnak helyére ugrás a Query Editor-ban
* Chekbox filtering: A jelölőnégyzetek használatával ideiglenesen kivehetjük a találati listából az egyes lekérdezéseket.



13. ábra - A Pattern registry-ből elérhető funkciók

### Result viewer, a lekérdezés-találat mutató

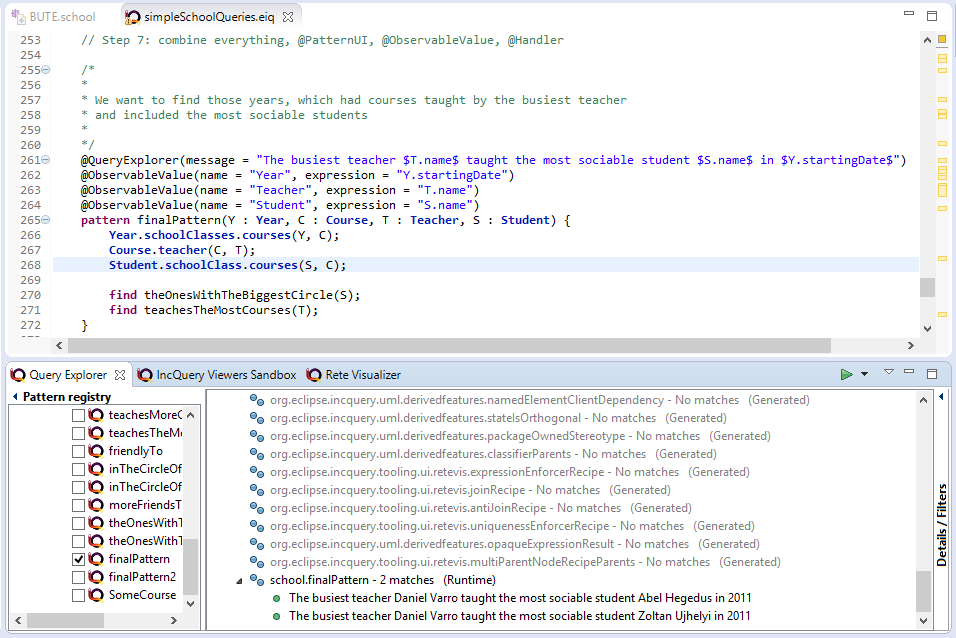
Ha legalább egy lekérdezés és egy példánymodell be van töltve, akkor a Query Explorer középső részén a lekérdezések eredményei egy fa struktúrában megjelennek. A legfelső szintű elem jelöli, hogy melyik példánymodell lett betöltve, és ez alatt találhatóak a lekérdezésekre a találatok.

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

* Pattern Registry
* Példánymodell
* Details/Filters szekció (lásd 8.2.5-ös fejezet)

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

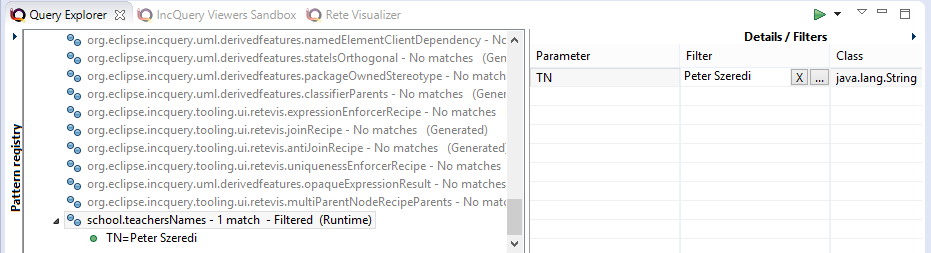
* @QueryExplorer: Ennek segítségével felparaméterezhetjük a lekérdezés szövegét, ahogy az a 14. ábrán is látszik.
* @ObservableValue: A Details/Filters panelben való megjelenés paraméterezhető fel vele.



14. ábra - EMF-IncQuery adatkötésre példa

### Details/Filters, a szűrő és részletező panel

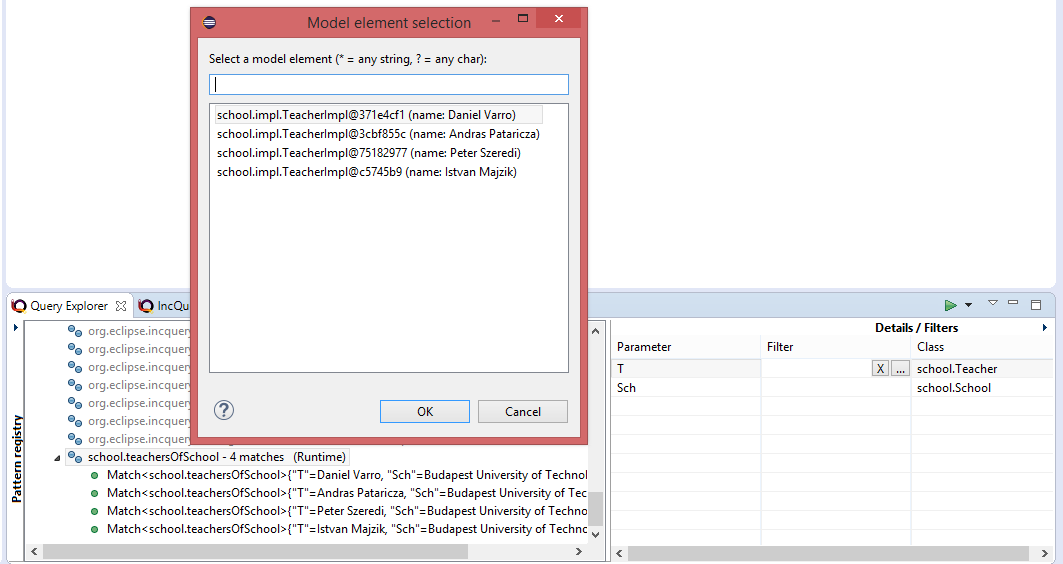
Ennek a panelnek a tartalma a kiválasztott elemtől függ. Ha a Result Viewer-ben a lekérdezés nevére kattintunk, akkor szűrőt állíthatunk be a találatokra. Amennyiben viszont egy szimpla találatra kattintunk, akkor pedig a találathoz tartozó egyes értékeket tekinthetjük meg részletezve itt.



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

A Details/Filters további funkciói:

* 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 feltételeknek megfelelő találatok azonnal megjelennek, a nem releváns adatok pedig eltűnnek.

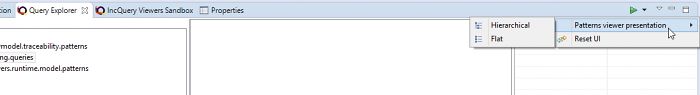


16. ábra - Felugró ablak a szűrési feltétel választásához a Details/filters nézetnél

### View menu

A képen látható, hogy a View menu hol helyezkedik el a Query Explorer nézeten belül. 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.

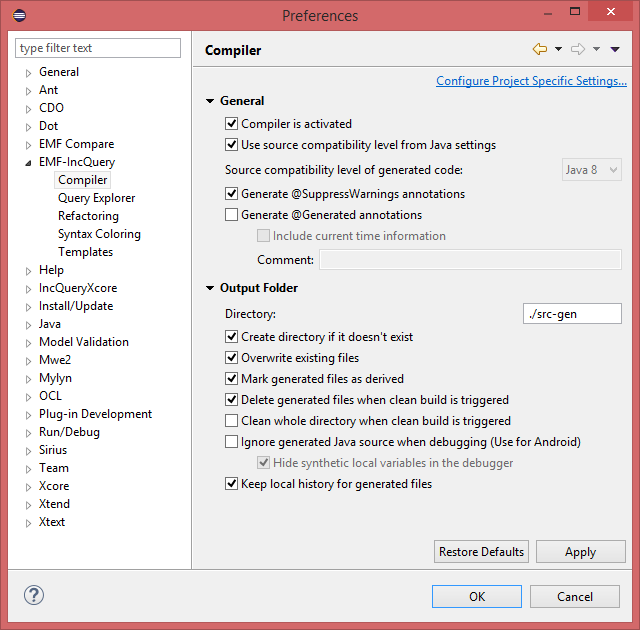


17. ábra - Lehetséges beállítások a View menu-ben

## Preferences, az EMF-IncQuery beállítási lehetőségei

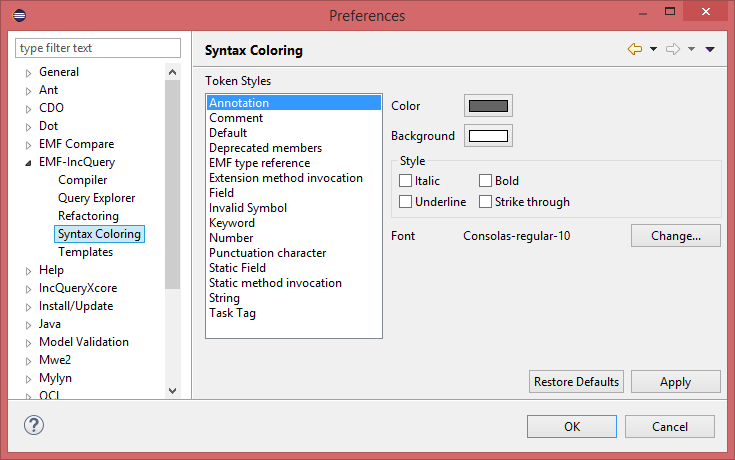
Mint szinte minden eclipse-be épülő alkalmazás, úgy az EMF-IncQuery is rendelkezik beállítási lehetőségekkel, ami a Window -> Preferences menüpontból érhető el. Itt lehetőségünk van globális vagy éppen csak projekt specifikus beállításokat megtennünk.

* Projekt specifikus beállítások:
  + Compiler beállításai
    - Általános: Itt lehet például ideiglenesen kikapcsolni a compiler-t ha gyorsítani akarunk az eszköz működésén.
    - Kimeneti mappa: A generált kóddal kapcsolatosan állíthatunk be dolgokat



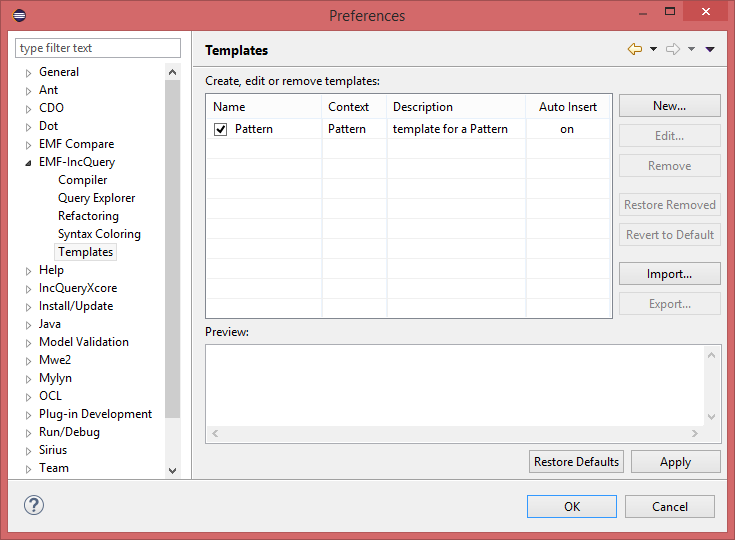
18. ábra – A compiler-re vonatkozó beállítások

* Globális beállítások
  + Query Explorer specifikus beállítások:
    - Wildcard mode ki/be kapcsolása: A lekérdezések indexelésével kapcsolatos beállítás.
    - Dynamic EMF mode ki/be kapcsolása: Abban az esetben érdemes bekapcsolni, ha ugyanazt a metamodellt különböző névvel, többször szeretnénk használni.
  + Query Editor specifikus beállítások:
    - Refactoring: Erőforrások elmentésével és átnevezésével kapcsolatos beállítási lehetőségek.
    - Syntax Coloring: A Query Editor formázási stílusát állíthatjuk be, mint például:
      * Szín
      * Háttér
      * Stílus
      * Betűtípus



19. ábra - Syntax Coloring beállításai

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



20. ábra - Templates beállításai

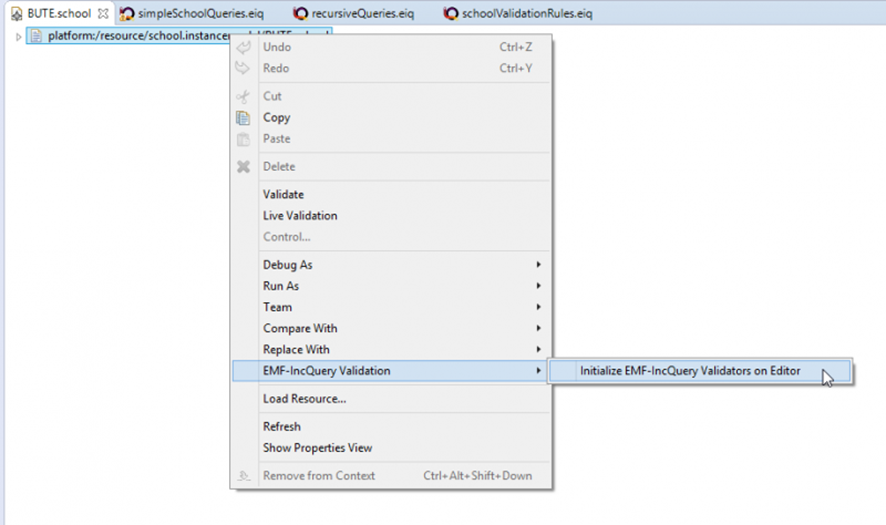
Mivel ezeknek a beállításoknak a végigtesztelése UI felületen elég hosszadalmas és nem is túl célravezető lenne, tesztelésemnek nem fő célja ezeknek a komponenseknek a teljes mélységig való lefedése, csak a fő funkciók meglétének ellenőrzése.

## Validation framework, az EMF-IncQuery validációs megoldása

Az EMF-IncQuery mintanyelve 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 a hibaüzenet is.

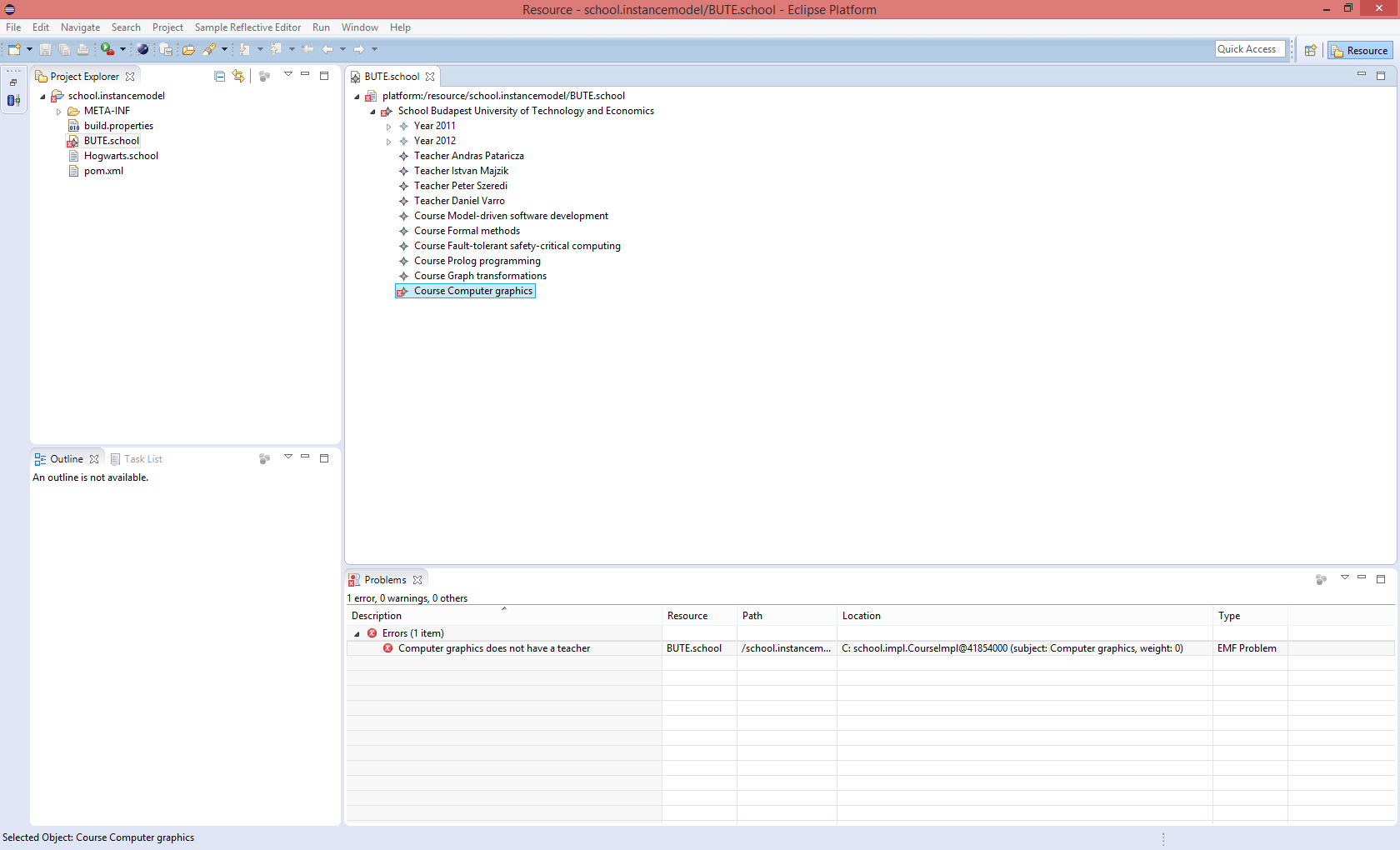
A következőkben a Validation framework használatát fogom bemutatni:

1. Először is, inicializálni kell a validátorokat a példánymodellen. Ez a 21. ábrán látható menüből érhető el. Ennek következtében létrejön egy .validation projekt is a workspace-ben.



21. 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 példánymodellbeli változtatások automatikusan megjelennek a Problems view-ban.
   2. A Problems view-ból lehetőség van a példánymodellbeli elemre navigálni, vagyis oda, ahol a változás történt (Jobb gomb -> Go to).
   3. Különböző adatokat érhetünk el a változásról a properties ablakból:
      1. leírás
      2. erőforrás
      3. típus
      4. hely
      5. elérési útvonal



22. ábra - Példánymodell változása esetén értesítés a problémáról

## Viewers, különböző nézeti megoldások

Az EMF-IncQuery Viewers 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.

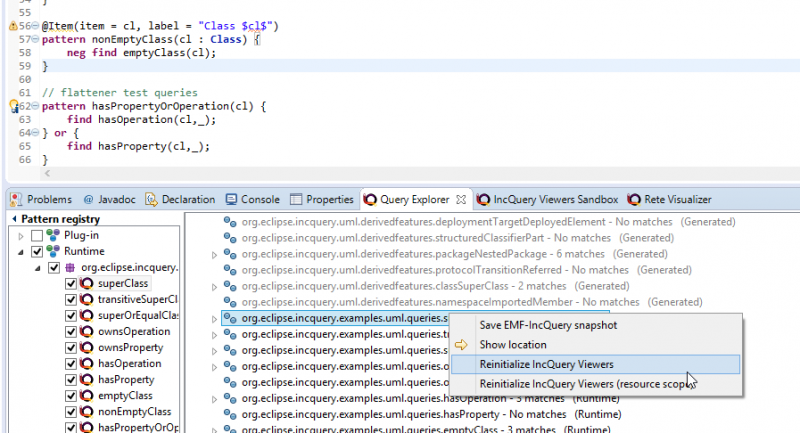
Ahhoz, hogy az ilyenfajta megjelenítés lehetséges legyen, a következő annotációkra van szükség a lekérdezések definiálásánál:

* @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

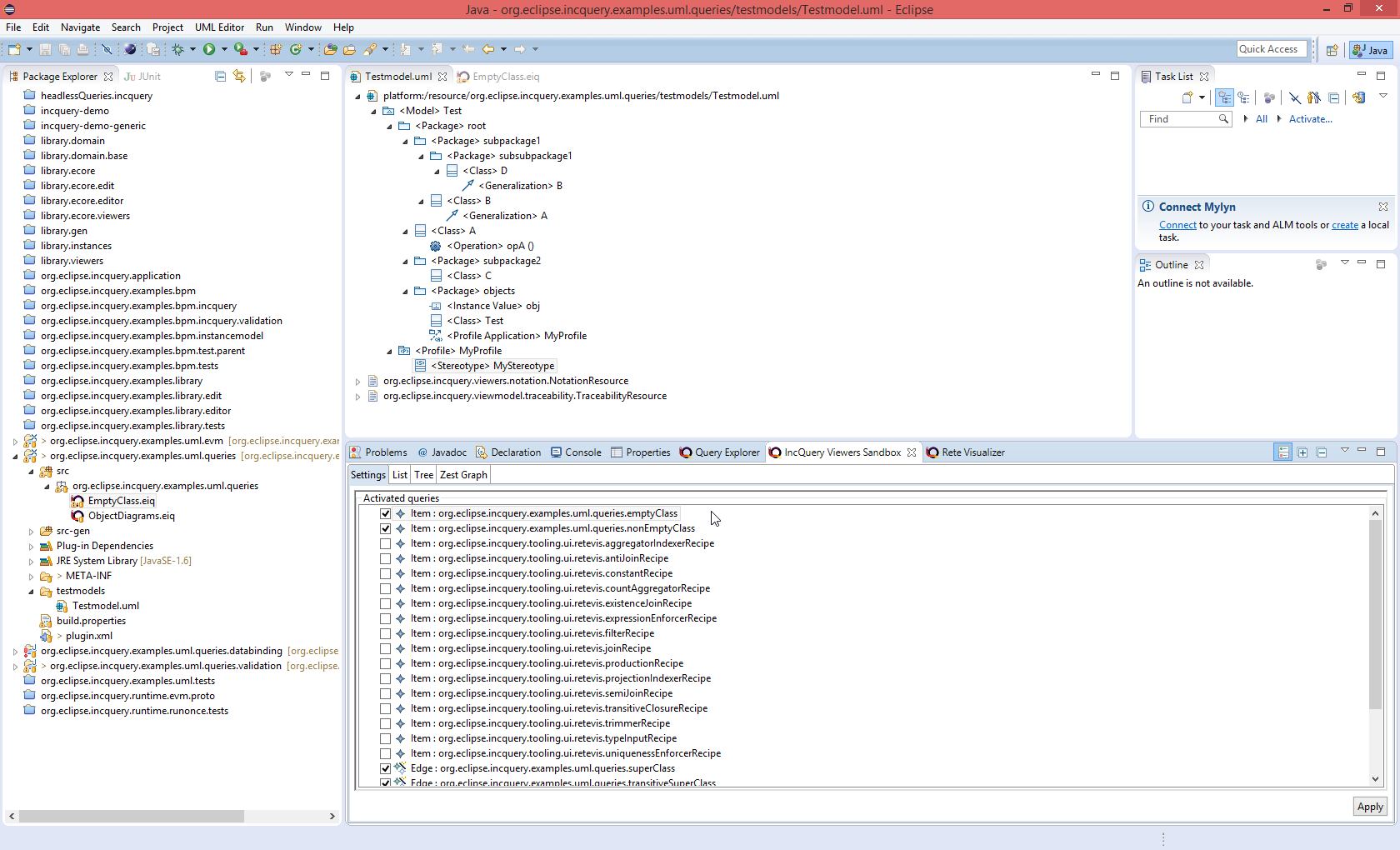
A következőkben a Viewers használatát fogom bemutatni:

* 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 23. ábrán látható módon tudunk inicializálni:



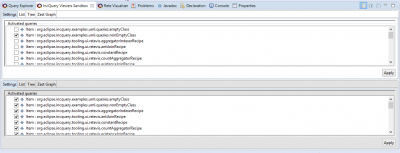
23. ábra - EMF-IncQuery Viewers inicializálása

* Ezután kezdődhet meg a nézetek konfigurálása:
  + Kiválaszthatjuk, hogy melyik lekérdezések eredményei jelenjenek meg a nézeteken:
    - a nézetek automatikusan szinkronizálódnak az Apply gombra kattintva



24. ábra - Lekérdezések kiválasztása a megjelenítéshez

* + Létrehozhatunk új komponenseket új nézetek számára.
  + Be is zárhatunk ilyen komponenseket.
  + Válthatunk horizontális vagy vertikális megjelenítés között.

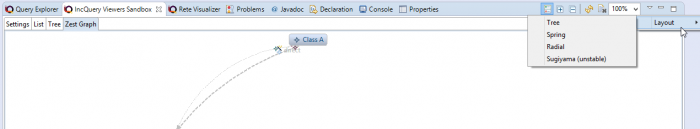


25. ábra - Horizontális és vertikális megjelenítés

Az elérhető nézetek tehát a következő képeken tekinthetőek meg. Lehetőség van lista-, fa- vagy éppen gráf-nézetben megtekinteni a találatokat.

|  |  |  |
| --- | --- | --- |
| List.png  26. ábra - ListViewer | Tree.png  27. ábra - TreeViewer | Zest.png  28. ábra - GraphViewer |

A GraphViewer-nél még további megjelenítési nézetek is elérhetőek. Ezeket csak itt találhatjuk meg, különböző gráftípusokat jelentek az elnevezések:



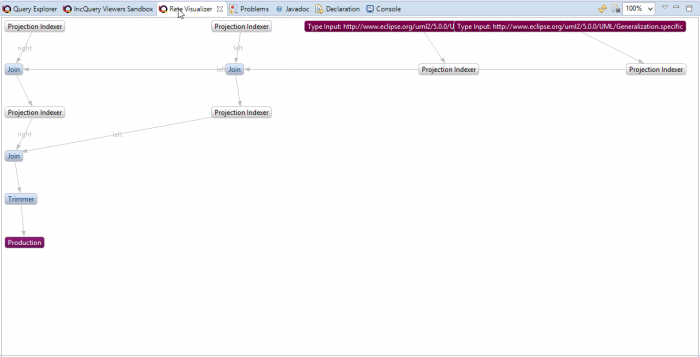
29. ábra - További elérhető nézetek GraphViewer esetén

## Rete Visualizer, a Rete háló megjelenítő

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

Használata:

* A 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.



30. ábra – A Rete Visualizer által megjelenített háló

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

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

## Testing framework, a tesztelő keretrendszer

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:



31. ábra - EMF-IncQuery snapshot készítése

A snaphot fájl elkészítésekor két lehetőségünk van:

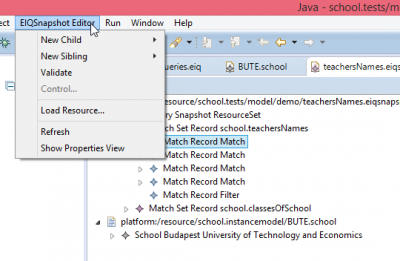
* Készítünk egy új .eiqsnapshot fájlt, amiben csak az aktuális lekérdezés eredménye szerepel példánymodellként (32. ábra).
* Vagy egy már létező snapshot fájlhoz adjuk hozzá a lekérdezésünk eredményét (33. ábra).

|  |  |
| --- | --- |
| Newsnapshot.png  32. ábra - Új snapshot fájl létrehozása | Existingsnapshot.png  33. ábra - Létező snapshot fájl használata |

### Az EIQ Snapshot Editor

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

* Create new child: Új gyermek elem létrehozása a példánymodellben.
* Create new sibling: Új testvér elem létrehozása a példánymodellben.
* Validate: A modell validációjának végrehajtása.
* Load resource: Például egy másik modell betöltésére ad lehetőséget.
* Refresh (model): A modell frissítése.
* Show properties view (még több információ, kontextusfüggő)
  + pattern qualified name
  + role
  + derived value
  + parameter name
  + value



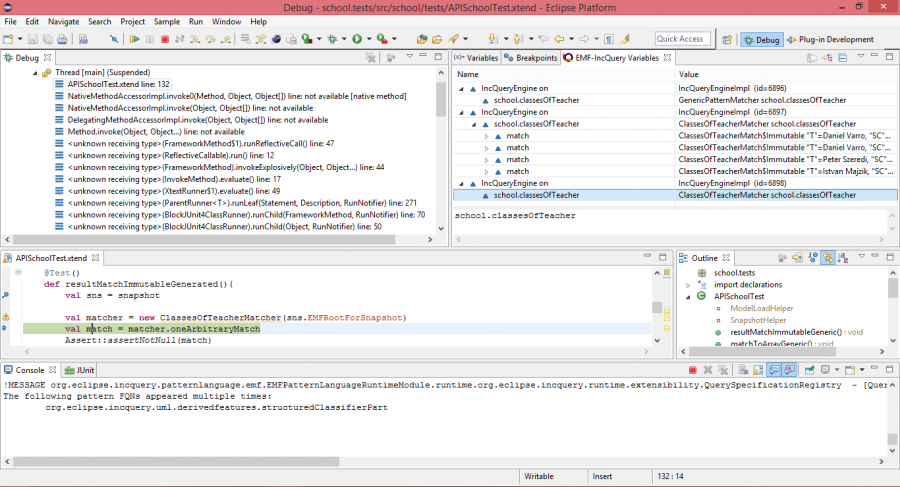
34. ábra - Az EIQSnapshot Editor kinézete

## 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.



35. ábra - EMF-IncQuery Debugger Tooling

# Tesztesetek felvétele az RCP Testing Tool segítségével

Ebben a fejezetben bemutatom a felhasználói felület use-case-einek lefedésére megalkotott teszteseteket, és az ehhez szükséges tervezői illetve optimalizációs lépéseket.

Mint azt az RCPTT bemutatásánál egy korábbi fejezetben már leírtam, az ilyen fajta teszteknél a legelső feladatunk a tesztelt alkalmazás, valamint a tesztelési kontextusok beállítása. A tesztelt alkalmazás, vagyis az AUT (Application Under Test) jelen esetben természetesen egy EMF-IncQuery-t tartalmazó eclipse, amiből a legújabb verziót, vagyis legújabb buildet letölthetjük az IncQuery honlapjáról.

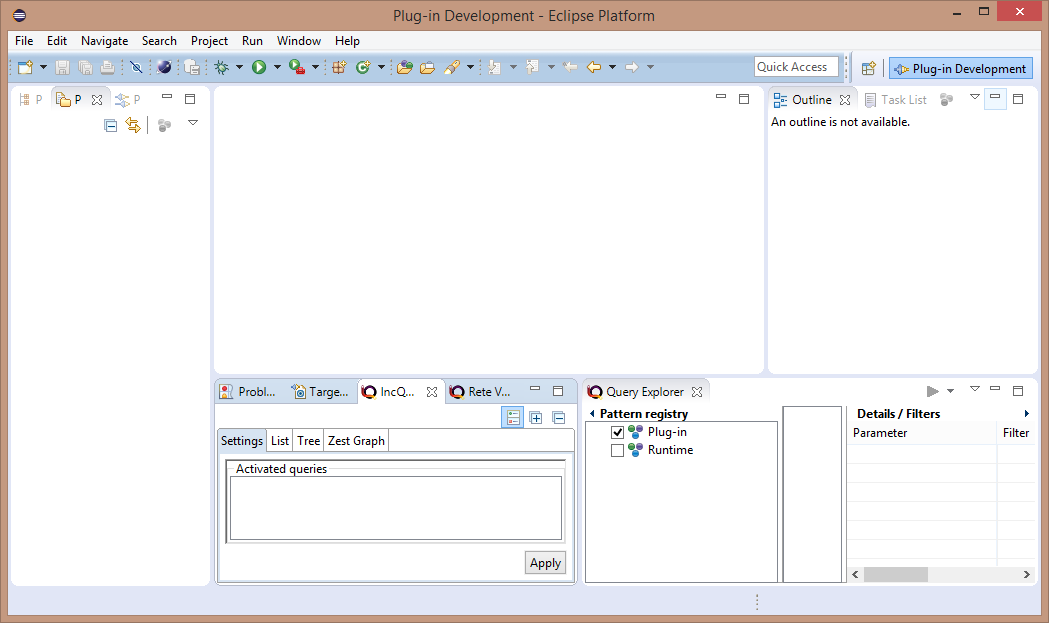
A tesztelési kontextus, ahogy azt az általános résznél is említettem, legalább 2 részből áll, a workbench és workspace definiálásából. Jelen esetben ezt még kibővítettem a legtöbb tesztesetnél egy kezdőállapotot kialakító szkripttel, hogy a tesztek között minél kisebb redundancia legyen.

## A workbench felvétele

Először tehát definiáltam egy, az EMF-IncQuery nézeteit tartalmazó workbench-et. Ehhez alap perspektívának a Plugin-Development perspektívát választottam, mert az EMF-IncQuery fejlesztéséhez is ezt használják, plusz még kiegészítetem a következő nézetekkel:

* Query Explorer
* Viewers Sandbox
* Rete Visualizer
* Project Explorer
* Problems View

Magyarázatra csak a Project Explorer és Problem View szorul, mivel a többire természetesen szükség van, ugyanis ezeket a komponenseket tesztelem. Előbbire azért van szükség, hogy a workspace-ben tudjak navigálni a teszteknél, ugyanis az IncQuery-nek nincs saját nézete a projektek megjelenítéséhez. A Problems View megléte pedig a tesztesetek felvételénél hasznos, ugyanis, így gyorsan észrevehetők a közben adódó esetleges hibák. Az összeállt workbench elemei jól láthatóak a 36. ábrán.



36. ábra - Az EMF-IncQuery-hez készült workbench

## A különböző workspace-ek felvétele

A különböző teszteseteknek különböző workspace-ekre van szükségük. Attól függően, hogy melyik teszt futtatása következik, 4 különböző kezdeti állapotot definiáltam:

* IncQuery School Workspace
* IncQuery School Queries Workspace
* IncQuery School Queries+Tests Workspace
* IncQuery UML Queries Workspace

Mint az a nevükből is sejthető, az UML és a School meta-és példánymodellt használó workspace-ek között a legnagyobb a különbség. A legtöbb teszt a School példát használja, az UML-es workspace-re csak az IncQuery Viewers és Rete Visualizer komponens miatt volt szükség, ugyanis az itteni lekérdezésekben találhatóak olyan annotációk, amik a grafikus megjelenítéshez szükségesek. A School Workspace csak az IncQuery projekt és lekérdezés létrehozásához szükséges komponenseket tartalmaz. Ezt egészíti ki a maradék kettő az előre definiált többfajta lekérdezéssel, valamint a Testing Framework, a teszteléséhez szükséges teszt projekttel.

A workspace-ek beállításainál fontos optimalizációs lépés volt a workspace törlésének kikapcsolása. Alapértelmezett esetben ugyanis minden egyes tesztnél, mindig újra létrejön, és a projektek buildelése is minden esetben megtörténik, ami mint kiderült egy teszt futtatásának leghosszabb része. Viszont fontos, hogy a különböző tesztek workspace-ei nem akadhatnak össze. Ezt úgy küszöböltem ki, hogy minden teszt csak a neki kellő projekteket tartja meg, így csak a teljesen különböző típusú workspace-t használó tesztek váltásánál tart tovább a buildelés és a tesztelés.

## A kezdeti állapotot beállító szkriptek felvétele

Mint azt a bevezetőben is említettem, a redundancia csökkentése miatt szükséges inicializáló szkriptek felvétele. Ennek köszönhetően, ha egy teszt ennél a kezdeti állapotot beállító résznél bukik el, akkor nem csak azt fogjuk látni, hogy az erre épülő tesztek elhasaltak valahol, hanem észre fogjuk venni, hogy az előkészítő lépéseknél történt a baj. Így ha a teszt utólagos karbantartására lesz szükség, normál esetben elegendő lesz csak ezt a szkriptet átírni.

A tesztek futtatásához a következő inicializációs lépésekre lehet szükség:

* Query Explorer kitisztítása
* IncQuery Viewer-ek inicializálása
* BUTE.school példánymodell betöltése
* empty-classes.uml példánymodell betöltése
* EmptyClass.eiq lekérdezések betöltése
* simpleSchoolQueries.eiq lekérdezések betöltése

Ezen felsorolt műveletek a legtöbbet használtak a tesztelés során, ezért gyűjtöttem ki őket egy külön szkriptbe. A Query Explorer törlésére szinte minden teszt előtt szükség van, enélkül problémák lehetnek egy teszt elvárt eredményének vizsgálatánál. A többi szkriptre csak a strukturáltság növelése és a redundancia csökkentése miatt volt szükség.

## Az elkészült tesztesetek

Az egyes tesztekből lehetőség van teszt sorozatok, úgynevezett Test Suite-ok kialakítására. Így az azonos UI komponenst tesztelő teszteket összefoghatjuk egy csokorba és könnyedén futtathatjuk külön a többitől.

Az elkészült „Test Suite”-ok ennek megfelelően a következők:

* IncQuery Preferences
* IncQuery Project Tests
* IncQuery Query Explorer Tests
* IncQuery Rete Visualizer
* IncQuery Testing Framework
* IncQuery Validation Framework
* IncQuery Viewers

Összesen 40 tesztet készítettem, ezeknek a legtöbbje a Query Explorer-hez készült. Itt egyedül csak az automatikus szinkronizáláshoz tartozó tesztesetet nem sikerült felvennem, amikor egy példánymodellen való változás azonnal megjelenik a Result Viewer-ben. Ennek az oka pedig az volt, hogy a példánymodellen való változtatást nem tudta megfelelően rögzíteni az RCPTT.

Ezen kívül még nem vettem fel a Validation Framework azon tesztjét, ahol a tesztelt eclipse-ből még egyet el kell indítani. Ez egyrészt nagyon erőforrás igényes - az eclipse képes összeomlani ennél a műveletnél -, másrészt az RCPTT nem is támogatja megfelelően újabb alkalmazások indítását a tesztelt eszközből.

A Debugger Tooling sem került tesztelésre, mert a tesztelés pillanatában még nem volt elérhető telepített alkalmazásként, csak a forráskódból fordítva. Így természetesen nem lett volna értelme tesztelni.

Az elkészült tesztesetek a függelékben olvashatóak, összegyűjtve róluk a legfontosabb információkat:

* Teszteset neve
* Használt workbench
* Használt workspace
* Használt inicializáló szkriptek
* Elvárt eredmény a megfelelő interakciók után

# Tesztesetek integrálása folytonos integrációs rendszerhez

Miután a tesztesetek elkészültek az RCPTT segítségével, már nem volt más hátra, csak ezeknek a teszteseteknek az integrálása egy build szerveren való futtatáshoz. Mivel az RCPTT ezt támogató része csak a szakdolgozat elkészítésének körülbelül felénél vált nyílt forráskódúvá, kezdetben egy másik megoldással próbálkoztam, ami támogatja a folytonos integrációt ezeknél a teszteknél. Miután elérhető lett az RCPTT teljes funkcionalitása, egyértelművé vált ennek a használata a továbbiakban. Ebben a fejezetben mindkét megoldást bemutatom, majd konklúziót vonok az egyes megoldások előnyeiről illetve hátrányairól.

## Az Apache Maven és a Jenkins

Az ilyen folytonos integrációt támogató rendszerek általában Apache Maven-t használnak a build folyamatok automatizálására. A Maven bevezette a POM (Project Object Model) fogalmát, ami egy buildelendő projektet ír le, annak minden függőségével. Ezek a projektekre vonatkozó függőségeket és egyéb műveleteket egy pom.xml fájlban definiálhatjuk, és hierarchikusan építhetjük fel az egymásra épülő modulokat.

A Maven képes a függőségeknek megfelelő komponenseket dinamikusan is letölteni és azokat eltárolni. A pom.xml fájlokban különböző célokat definiálhatunk a Maven számára, amit az automatikusan végrehajt. Ilyen célok lehetnek például a forráskód fordítása, tesztek futtatása, .jar fájlok készítése. Ezek a célok szintén épülhetnek egymásra, vagyis meghatározhatjuk, hogy egy cél végrehajtásához egy másiknak már teljesülnie kellett előtte. Így komplex build folyamatokat rakhatunk össze. A Maven használata történhet parancssorból vagy Linux terminálból, esetleg fejlesztőeszközbe telepített plugin segítségével, ami az eclipse-hez is elérhető.

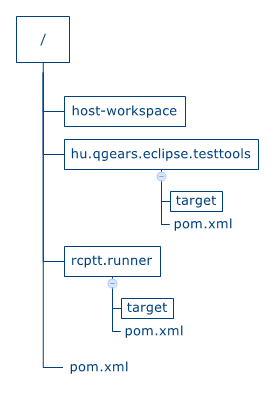
A folyamatos integráció egy másik fontos eszköze, az úgynevezett Jenkins integrációs eszköz. Ez egy szerver alapú rendszer, így egy webes felületen érhetjük el telepítés után. A Jenkins támogatja a legtöbb verziókezelő rendszert, mint a Cvs, Subversion, Git, és szorosan együtt tud működni Maven alapú projektekkel. Lényegében Maven alapú projektek céljainak automatikus végrehajtására használhatjuk. Ezzel az eszközzel folyamatosan követhetjük a projekteink állapotát, a különböző telepíthető plugin-ek telepítésével plusz funkciókat érhetünk el, mint például kód-fedettségi adatok megjelenítése és statisztika felállítása. Az egyes buildek futtatására különböző feltételeket állíthatunk be, mint például, hogy milyen sűrűn fussanak a buildek, vagy esetleg csak a verziókezelő rendszerben történt változás esetén történjen buildelés. Az RCPTT tesztek integrálásához is ezeket az eszközöket használtam, ezért volt szükség a bemutatásukra.

## Külső megoldás használata a tesztek integrálásához

Mielőtt az RCPTT nyílt forráskódúvá vált volna, egy külső megoldást használtam a tesztek folytonos integrációs környezetbe való helyezéséhez. A GitHub-on fellelhető projekt segítségével lehetőségünk van a következőkre:

* Rendszer-változóként megadott eclipse launch konfiguráció elindítása.
* Különböző eclipse parancsok végrehajtása.
* RCPTT AUT (a tesztelt alkalmazás) elindítása és futtatása.
* JUnit XML alapú teszt jelentés készítése a végrehajtott teszt sorozathoz.

Ez a megoldás is a fentebb említett Maven-t használja a buildeléshez és a tesztek futtatásához, kiegészítve specifikus, az eclipse-t futtató parancsokkal. Ahhoz, hogy a tesztelés automatikusan működjön, a GitHub-on található hu.qgears.eclipse.testtools projekt mellé létre kellett hoznom egy üres, rcptt.runner nevű projektet, ami az RCPTT tesztek futtatásáért felelős. Ennek megfelelően a következő projekt hierarchiát alakítottam ki, a legfontosabb elemeket feltüntetve:



37. ábra – Projekt hierarchia külső megoldás használatával

A saját rcptt.runner projektem kialakításában, csak a pom.xml tartalmának volt szerepe, amelyben be kellett állítani a következő property-ket:

* Indigo P2 tárhelyhez URL: A függőségek feloldásához szükséges komponenseket tartalmaz ez a repository.
* RCPTT P2 tárhelyhez URL: Az RCPTT elindításához szükséges.
* Testtools projekt P2 tárhelyként csomagolva: Ezt a testtools projekt automatikusan megteszi a target mappájába.
* Host-workspace elérési útvonal: Szükséges megadnunk a teszteket tartalmazó workspace helyét is, ugyanis itt fogja keresni a teszteléshez használt teszt sorozatot, vagyis Test Suite-ot (esetünkben ez az IncQuery\_RCPTT\_Tests nevet kapta). Ezeket én a 37. ábrán látható host-workspace mappában helyeztem el.
* Jelentés helye: A kimeneti XML alapú fájlok helye.

Ezek az értékek a futtatáskor átadódnak a különböző eclipse parancsoknak paraméterül, és így következik be az elvárt működés.

A két különálló modul összefogása miatt szükség volt még a gyökér mappában található szülő pom.xml fájlra, ami a két modult lénygében importálja a következő módon:

<modules>

<module>hu.qgears.eclipse.testtools</module>

<module>rcptt.runner</module>

</modules>

Ezután a gyökér mappában a Maven mvn clean install parancsát hívva, ami először törli a generált fájlokat, majd végrehajtja a pom.xml fájlokban meghatározott célokat és műveleteket, elindul egy RCPTT alkalmazás, ami elindít egy AUT-ot és végrehajtja a teszteket. Ennek befejeztével, pedig leállítja az eclipse-eket és elkészíti a kimeneti XMLfájlt az rcptt.runner target mappájában, ami tartalmazza, hogy az egyes tesztek sikeresek voltak-e.

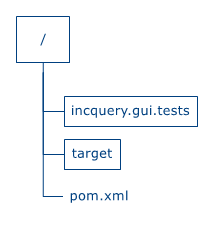
Fontos megjegyezni, hogy a pom.xml fájlokban nem lehet definiálni az AUT elérési útvonalát, annak ott kell lennie induláskor az RCPTT programban. Ez a tesztek első futtatásánál kényelmetlen lehet.

## Az RCPTT Runner és Jenkins használata a tesztek integrálásához

Miután az RCPTT teljesen nyílt forráskódú lett, már trükközés és külső megoldás nélkül lehetségessé vált a tesztek automatikus futtatása.

### Az RCPTT Runner bemutatása és használata

Megjelent az RCPTT Maven plugin, ami képes automatikusan letölteni az előző megoldásban levő rcptt.runner-hez hasonló elven működő RCPTT Runner-t. Ez szintén parancssoros argumentumok átadásával működik, de ezt elrejti előlünk a plugin. Mindössze egy pom.xml-t kell definiálnunk a megfelelő paraméterezéssel a projekt könyvtárban, majd az mvn clean package paranccsal futtatni a teszteket, aminek a végén a target mappában megjelennek az eredmények. A teszteket automatikusan megkeresi a pom fájlal egy szinten levő mappákban vagy almappákban, így egy egyszerű projektszerkezetet kaphatunk.



38. ábra - Projekt hierarchia az RCPTT Runner és Maven plugin használatával

Esetünkben az incquery.gui.tests mappában helyezhetjük el a tesztprojektünket, a target könyvtárban pedig az eredményt találhatjuk, ami itt nem csak egy JUnit XML alapú fájl, hanem egy HTML alapú jelentést is kapunk screenshot-okkal az utolsó állapotról, ha esetleg elbukott egy teszt. Ezen kívül különböző log fájlokat is találhatunk a kimeneti mappában, az RCPTT Runner-re vagy az AUT-ra vonatkozóan, ami jól jön a problémák elhárításánál.

Ahhoz, hogy az automatikus futtatás működhessen, szükség volt a pom.xml-ben az AUT és a Runner elérési útjának definiálására, amit explicit elérési úttal adtam meg:

<properties>

<autPath>/usr/eclipse-IncQuery-AUT/</autPath>

<runnerPath>/usr/rcptt.runner-incubation-1.5.6-N201504182315.zip</runnerPath>

</properties>

<aut>

<explicit>${autPath}</explicit>

</aut>

<runner>

<explicit>${runnerPath}</explicit>

</runner>

Kezdetben problémákba ütköztem a tesztek futtatásával, ugyanis a korábban is említett workspace – az UML-ről School példára való – váltás miatt timeout-olt néhány teszt. A kimeneti log fájlokban a nem elegendő memóriára utaló üzenetek jelentek meg, ezért megemeltem az AUT által használható memóriát valamint a tesztek maximális futtatási idejét:

<aut>

<explicit>${autPath}</explicit>

<vmArgs>

<vmArg>-Xmx1024m</vmArg>

<vmArg>-XX:MaxPermSize=512m</vmArg>

</vmArgs>

</aut>

<testOptions>

<execTimeout>7200</execTimeout>

<testExecTimeout>600</testExecTimeout>

</testOptions>

Ezután a mvn clean package parancs futtatásával, már az összes teszt lefutott a helyi gépemen, nem volt már más hátra, csak a Jenkins bekonfigurálása a tesztek automatikus futtatására.

### A Jenkins konfigurálása

Egyszerűbb esetben a Jenkins konfigurálása csak annyiból áll, hogy megadjuk a verziókövető rendszert, amit használni akarunk, valamint, hogy milyen Maven parancsot adjon ki a futtatás során.

Mivel jelen esetben UI felületet tesztelek, a futtatás közben szükség van a háttérben a felhasználói felületek kezelésére. Ehhez az Xvnc-t használtam, amihez letölthető Jenkins plugin is. Segítségével egy operációs rendszeren belül több applikációhoz lehet felhasználói felületet biztosítani. Ahhoz, hogy a használt felület meghatározható legyen, még szükség volt a Metacity ablakkezelő telepítésére, ami a Linux GNOME felületet használja, így Jenkins-en való futtatás során is kaphatunk screenshot-okat az esetleg elhasaló teszteknél.

A tesztjeimet feltöltöttem a GitHub verziókezelő rendszerbe, hogy a Jenkins onnan tudja elérni őket. Így a teszteken való módosításokat könnyedén tudtam ellenőrizni a Jenkins-en való futtatással. Ezzel kapcsolatban egy megoldandó probléma volt, hogy a GitHub-ról behúzott tesztek Jenkins-en való futtatásakor a .git – GitHub adminisztrációs adatokat tartalmazó – mappa miatt a tesztelt alkalmazásban a projektek elnevezései kiegészültek a tárhely nevével, így az RCPTT szkriptek már nem jól hivatkoztak a projektek neveire. Ezt kiküszöbölendő, fel kellett venni egy, a .git mappát kitörlő parancsot a Jenkins konfigurációba, még a Maven parancs végrehajtása elé. Ezután már futottak a tesztek.

Az utolsó megoldandó probléma a kód-fedettségi adatok kinyerése illetve megjelenítése volt. Ehhez a JaCoCo Jenkins plugin-t használtam, ami mindössze egy futási adatokat tartalmazó jacoco.exec fájlt igényel valamint, hogy hol található a fordított forráskód. Ezekből az adatokból képes kód-fedettségi adatok megjelenítésére, valamint az egymás után következő buildek fedettségi adataiból statisztika megjelenítésére.

A jacoco.exec fájl létrehozásához fel kellett vennem a pom.xml-ben az AUT VM argumentumai közé még egy sort:

<vmArg>-javaagent:/usr/jacoco/lib/jacocoagent.jar=destfile=/var/lib/jenkins/jobs/EMF-IncQuery-RCPTT-GUI-Tests/workspace/target/jacoco.exec

</vmArg>

Ehhez le kellett töltenem természetesen a JaCoCo-t a megadott helyre, de ezután már létrejött a jacoco.exec fájl, amit át lehetett adni a Jenkins Plugin-nek.

A fordított forráskód jelen esetben az AUT fájljait jelentette, ami viszont alapesetben, az RCPTT által nem másolódik a target mappába. Ez problémát jelent, ugyanis, a JaCoCo Jenkins plugin-nek nem adható meg abszolút elérési útvonal. Ezért még a fedettségi adatok kiértékelése előttre felvettem shell parancsokat, amik a target mappába másolják az AUT-ot, és így már megjelentek a kód-fedettségi adatok.

## Konklúzió

A két megoldást összehasonlítva, az első mellett nem szólnak érvek. A második megoldás összerakása könnyebb és egyszerűbb, valamint a kapott kimenet is sokkal informatívabb a HTML alapú eredményeknek és log fájloknak köszönhetően, amik az első megoldásnál nem álltak rendelkezésre.

A második megoldást használva a 40 db teszt lefut közel 20 perc alatt és ekkor már a kód-fedettségi adatok generálása is megtörténik, ami véleményem szerint elfogadható idő. Ennek elérésében sokat segített, hogy a teszteléshez használt workspace, csak abban az esetben törlődik, ha az aktuális teszteset másik példánymodellt használ. Ennek hátránya, hogy oda kell figyelnünk rá, hogy a tesztesetek ne módosítsák úgy a workspace-t, hogy az más tesztre kihatással lenne. Jelen tesztek esetében ilyen nem történik. Ha workspace törlését választanám minden egyes teszt előtt, az drasztikusan megnövelné a futási időt, úgy akár órákig is eltarthatna az összes teszt futása.

A kód-fedettségi adatokat vizsgálva, jól látszik, hogy az egész eszköz közel 40%-át lefedik a tesztesetek:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **utasítás** | **elágazás** | **komplexitás** | **sor** | **metódus** | **osztály** |
| 43%  **M[[5]](#footnote-5):** 269769  **C[[6]](#footnote-6):** 200782 | 23%  **M:** 31569  **C:** 935 | 27%  **M:** 28090  **C:** 10513 | 41%  **M:** 60656  **C:** 42030 | 46%  **M:** 9452  **C:** 7970 | 73%  **M:** 625  **C:** 1674 |

5. táblázat - A teljes eszköz fedettségi adatai

Kigyűjtöttem az egyes UI komponensek fedettségi adatait is. Itt jól látszik, hogy komponensenként körülbelül 80%-os lefedettséget sikerült elérni, és az összes ehhez tartozó osztály használva volt. Ez nagyon jó eredménynek tekinthető. A \*. patternlanguage.ui komponens főként a Query Editor-hoz és annak beállításaihoz tartozik, ami nem lett részletesebben tesztelve, ennek köszönhető a 0%-os lefedettség. Megfigyelhetőek még néhány komponensnél a 0%-os elágazási lefedettségek. Ez csupán azt jelenti ebben az esetben, hogy nem voltak elágazások az adott komponensben.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **komponens neve** | **utasítás** | **elágazás** | **komplexitás** | **sor** | **metódus** | **osztály** |
| \*.patternlanguage.emf.ui | 82% | 41% | 80% | 84% | 94% | 100% |
| \*.patternlanguage.ui | 0% | 0% | 0% | 0% | 0% | 0% |
| \*.testing.ui | 74% | 0% | 83% | 73% | 83% | 100% |
| \*.tooling.generator.model.ui | 83% | 50% | 86% | 90% | 100% | 100% |
| \*.tooling.ui | 83% | 0% | 75% | 78% | 75% | 100% |
| \*.validation.runtime.ui | 84% | 50% | 43% | 75% | 75% | 100% |
| \*.viewers.tooling.ui | 52% | 0% | 60% | 60% | 60% | 100% |

6. táblázat - Az eszköz GUI komponenseinek fedettségi adatai

# Összefoglalás és 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 11](#_Toc419141452)

[2. ábra - A Jnario-s tesztkeretrendszer felépítése 19](#_Toc419141453)

[3. ábra - Kódfedettség *Jnario* és *SWTBot* használatával 20](#_Toc419141454)

[4. ábra - Az RCP Testing Tool kinézete 24](#_Toc419141455)

[5. ábra - Az *RCP Testing Tool* *Control Panel* 25](#_Toc419141456)

[6. ábra - Tesztek futtatása az *RCPTT*-vel 25](#_Toc419141457)

[7. ábra - Részlet a Sonar kód-fedettségi adatokból 32](#_Toc419141458)

[8. ábra - A *Query Editor* kinézete 34](#_Toc419141459)

[9. ábra - A *Query Explorer* kinézete 35](#_Toc419141460)

[10. ábra - *Pattern* regisztrálása a *Query Explorer*-ben 35](#_Toc419141461)

[11. ábra - *Magic Green Button* állapota megnyitott példánymodell esetén 36](#_Toc419141462)

[12. ábra - "Magic Green Button" állapota megnyitott .eiq fájl esetén 37](#_Toc419141463)

[13. ábra - A pattern registry-ből elérhető funkciók 38](#_Toc419141464)

[14. ábra - EMF-IncQuery adatkötésre példa 39](#_Toc419141465)

[15. ábra - Szűrési feltétel megadása a Details/Filters-ben 39](#_Toc419141466)

[16. ábra - Felugró ablak a szűrési feltétel választásához a Details/filters nézetnél 40](#_Toc419141467)

[17. ábra - Lehetséges beállítások a View menu-ben 40](#_Toc419141468)

[18. ábra - Compiler-re vonatkozó beállítások 41](#_Toc419141469)

[19. ábra - Syntax Coloring beállításai 42](#_Toc419141470)

[20. ábra - Templates beállításai 43](#_Toc419141471)

[21. ábra - EMF-IncQuery validátorok inicializálása 44](#_Toc419141472)

[22. ábra - Példánymodell változása esetén értesítés a problémáról 45](#_Toc419141473)

[23. ábra - EMF-IncQuery Viewers inicializálása 46](#_Toc419141474)

[24. ábra - Lekérdezések kiválasztása a megjelenítéshez 46](#_Toc419141475)

[25. ábra - Horizontális és vertikális megjelenítés 47](#_Toc419141476)

[26. ábra - ListViewer 47](#_Toc419141477)

[27. ábra - TreeViewer 47](#_Toc419141478)

[28. ábra - GraphViewer 47](#_Toc419141479)

[29. ábra - Rete Visualizer által megjelenített háló 48](#_Toc419141480)

[30. ábra - EMF-IncQuery snapshot készítése 49](#_Toc419141481)

[31. ábra - Új snapshot fájl létrehozása 50](#_Toc419141482)

[32. ábra - Létező snapshot fájl használata 50](#_Toc419141483)

[33. ábra - Az EIQSnapshot Editor 51](#_Toc419141484)

[34. ábra - EMF-IncQuery Debugger Tooling 51](#_Toc419141485)

[35. ábra - Az EMF-IncQuery-hez készült Workbench 53](#_Toc419141486)

[36. ábra – Projekt hierarchia külső megoldás használatával 57](#_Toc419141487)

[37. ábra - Projekt hierarchia az RCPTT Runner és Maven plugin használatával 59](#_Toc419141488)

# Táblázatok jegyzéke

[1. táblázat. Példa táblázat feliratára 11](#_Toc419141489)

[2. táblázat - *SWTBot*, *WindowTester Pro* és *Jubula* összehasonlító táblázat[8] 22](#_Toc419141490)

[3. táblázat - EMF-IncQuery tesztkatalógus 1. rész 30](#_Toc419141491)

[4. táblázat - EMF-IncQuery tesztkatalógus 2. rész 31](#_Toc419141492)

[5. táblázat - A teljes eszköz fedettségi adatai 62](#_Toc419141493)

[6. táblázat - Az eszköz GUI komponenseinek fedettségi adatai 62](#_Toc419141494)

# 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 [http://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/](http://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/FeatureSetAndTesting/QueryExplorer)
6. Szabó Rudi, *Eclipse alkalmazások felületének automatikus tesztelése* <https://diplomaterv.vik.bme.hu/hu/Theses/Eclipse-alkalmazasok-feluletenek-automatikus>
7. Sas Csaba, *Az EMF-IncQuery felhasználói felületének automatikus tesztelése* <https://diplomaterv.vik.bme.hu/hu/Theses/Az-EMFIncQuery-felhasznaloi-feluletenek>
8. Koska Sándor, *Eclipse alapú alkalmazások felhasználói felület tesztelése* <https://diplomaterv.vik.bme.hu/hu/Theses/Eclipse-alapu-alkalmazasok-felhasznaloi>
9. Ficsor Lajos, Dr. Kovács László, Krizsán Zoltán, Dr. Kusper Gábor, *Szoftvertesztelés* <http://www.inf.unideb.hu/kmitt/konvkmitt/szoftverteszteles/book.xml.html#id490923>
10. Wikipedia – Apache Maven <http://hu.wikipedia.org/wiki/Apache_Maven>
11. Wikipedia – Jenkins <http://hu.wikipedia.org/wiki/Jenkis>
12. hu.qgears.eclipse.testtools projekt a GitHub-on https://github.com/qgears/testtools

# Függelék

A 40 teszteset a következőkben olvasható strukturált, táblázatos formában:

|  |  |
| --- | --- |
| Teszteset neve | Create IncQuery Project |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Workspace |
| Inicializáló szkriptek | - |
| Várt eredmény | IncQuery projekt létrejöttének ellenőrzése, létrejövő src és src-gen mappák üres-e. |

|  |  |
| --- | --- |
| Teszteset neve | Create Pattern |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Workspace |
| Inicializáló szkriptek | - |
| Várt eredmény | Pattern fájl létrejöttének ellenőrzése, a fájl a varázslóban begépelt szöveget tartalmazza. |

|  |  |
| --- | --- |
| Teszteset neve | Create .eiqgen |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Workspace |
| Inicializáló szkriptek | - |
| Várt eredmény | A megfelelő helyen a generátor fájl létrejött. A függőségek között megjelent a „school”. |

|  |  |
| --- | --- |
| Teszteset neve | Pattern Autocomplete |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Workspace |
| Inicializáló szkriptek | - |
| Várt eredmény | Automatikus kiegészítés tesztelése. A tesztben elvárt megjelenő ajánlások: „courses, homeroomedClass, name, school”. |

|  |  |
| --- | --- |
| Teszteset neve | Preferences - Compiler |
| Workbench | IncQuery Workbench |
| Workspace | - |
| Inicializáló szkriptek | - |
| Várt eredmény | A Preferences-ben elérhetőek a Compiler beállításai. |

|  |  |
| --- | --- |
| Teszteset neve | EMF-IncQuery exists in preferences |
| Workbench | IncQuery Workbench |
| Workspace | - |
| Inicializáló szkriptek | - |
| Várt eredmény | A Preferences-ben megtalálhatóak az EMF-IncQuery beállításai. |

|  |  |
| --- | --- |
| Teszteset neve | Preferences – Query Explorer |
| Workbench | IncQuery Workbench |
| Workspace | - |
| Inicializáló szkriptek | - |
| Várt eredmény | A Preferences-ben elérhetőek a Query Explorer beállításai. |

|  |  |
| --- | --- |
| Teszteset neve | Preferences – Refactoring |
| Workbench | IncQuery Workbench |
| Workspace | - |
| Inicializáló szkriptek | - |
| Várt eredmény | A Preferences-ben elérhetőek a Refactoring beállításai. |

|  |  |
| --- | --- |
| Teszteset neve | Preferences - Syntax Coloring |
| Workbench | IncQuery Workbench |
| Workspace | - |
| Inicializáló szkriptek | - |
| Várt eredmény | A Preferences-ben elérhetőek a Syntax Coloring beállításai. |

|  |  |
| --- | --- |
| Teszteset neve | Preferences - Templates |
| Workbench | IncQuery Workbench |
| Workspace | - |
| Inicializáló szkriptek | - |
| Várt eredmény | A Preferences-ben elérhetőek a Templates beállításai. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Details/Filters - Filtering |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Szűrés előtti 4 találat ellenőrzése, utána 1 találat ellenőrzése a school.teacherNames lekérdezésre tekintettel. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Details/Filters - Popup window |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | A szűrési feltétel felugró ablak 5 elemet tartalmaz a school.studentOfSchool lekérdezésre tekintettel. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Magic Green Button - Load as model |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | A simpleSchoolQueries.eiq lekérdezések modellként való betöltődésének vizsgálata. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Magic Green Button - Load pattern(s) |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | A simpleSchoolQueries.eiq lekérdezések Pattern Registry-be való betöltésének vizsgálata. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Magic Green Button - Load ResourceSet |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | BUTE.school példánymodell betöltődésének vizsgálata. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Pattern Registry - Checkbox filtering |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | A school.schools lekérdez vizsgálata, hogy egy találat van-e rá. Majd a checkbox kitörlése és annak vizsgálata, hogy üres lett-e. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Pattern Registry - Plugins can not unloaded |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Egy plugin-ra kattintva annak kiregisztrálásának megkísérlése, és sikertelen kiregisztrálás vizsgálata. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Pattern Registry - Show location |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Pattern-en a „Show location”-re kattintva annak vizsgálata, hogy a kijelölt szöveg a pattern neve lett a .eiq fájlban, vagyis odaugrott a kurzor. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Pattern Registry - Unregister pattern |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Összes pattern kiregiszrálása, majd annak a vizsgálata, hogy 0 maradt-e betöltve. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Register patterns in Query Explorer |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | Pattern regisztrálása a .eiq fájlon jobb gombbal kattintva, és annak a vizsgálata, hogy mind a 31 pattern betöltődött-e a „Pattern Registry”-be. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Register patterns in Query Explorer (Right clicked in editor) |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | A Query Editor-ban jobb gombbal kattintva a pattern-ek regisztrálása, majd annak a vizsgálata, hogy mind a 31 pattern betöltődött school néven és be van jelölve a jelölőnégyzet. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Result Viewer - Data binding |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Adatkötés helyességének ellenőrzése, mind a .eiq fájlban, mind a találatban a megfelelő szöveg ellenőrzése. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Result Viewer - Query Results |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | Példánymodell és lekérdezés betöltése, majd egy találat vizsgálata, hogy van 1 egyezés. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Result Viewer - Show location (to .eiq file) |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Show location navigálás helyességének ellenőrzése a .eiq fájlba a lekérdezés nevén kattintva jobb gombbal.. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Result Viewer - Show location (to instance model) |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Show location navigálás helyességének ellenőrzése a példánymodell felé, a lekérdezés találatának nevén kattintva jobb gombbal. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - Result Viewer - Unload instance model |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Példánymodell kiregisztrálásának ellenőrzése. Kiregisztrálás előtt 1 elem a Result Viewer-ben, utána 0 elem ugyanitt. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - View menu - Change package representation |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Package hierarchia változtatásának ellenőrzése a Pattern Registry-ben. Flat esetén 5 fia van a plugin szekciónak, hierarchical esetben pedig csak 1. |

|  |  |
| --- | --- |
| Teszteset neve | Query Explorer - View menu - Reset UI |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Query Explorer törlésének a vizsgálata. Először 1 elemet tartalmaz a Result Viewer, a törlés után 0-t. |

|  |  |
| --- | --- |
| Teszteset neve | Rete Visualizer - Clear graph |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq) |
| Várt eredmény | Rete gráf törlésének ellenőrzése, először 13 csomópont, törlés után pedig 0 csomópont a nézetben. |

|  |  |
| --- | --- |
| Teszteset neve | Rete Visualizer - Create rete |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq) |
| Várt eredmény | Rete gráf létrehozásának vizsgálata, először 0 csomópont, a létrehozás után viszont 13 csomópont a nézetben. |

|  |  |
| --- | --- |
| Teszteset neve | Rete Visualizer - Testing layouts |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq) |
| Várt eredmény | Rete gráf különböző megjelenítési fajtáinak a tesztelése, gráf típusának a lekérése minden egyes esetben. |

|  |  |
| --- | --- |
| Teszteset neve | Viewers - Sandbox - Create new component |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq)  Initialize viewers |
| Várt eredmény | Új komponens készítése a Viewers Sandbox-ban és ennek a létrejöttének a vizsgálata. |

|  |  |
| --- | --- |
| Teszteset neve | Viewers - Sandbox - Initialize |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq) |
| Várt eredmény | Viewers Sandbox inicializálásának ellenőrzése, kezdetben 0 aktivált lekérdezés. Az inicializálás után megtalálható a Sandbox-ban az aktivizált lekérdezés. |

|  |  |
| --- | --- |
| Teszteset neve | Viewers - Sandbox - View synchronization |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq)  Initialize viewers |
| Várt eredmény | Automatikus szinkronizáció vizsgálata, a Settings-ben változtatások végzése, majd az ennek megfelelően változó elemek vizsgálata az egyes View-kban. |

|  |  |
| --- | --- |
| Teszteset neve | Viewers - Sandbox - Zest Graph (Clear graph) |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq)  Initialize viewers |
| Várt eredmény | A Graph Viewer törlésének vizsgálata. Kezdetben 3 elem a View-ban, a törlés után viszont már 0 elem található ott. |

|  |  |
| --- | --- |
| Teszteset neve | Viewers - Sandbox - Zest Graph (Testing layouts) |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery UML QueriesWorkspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (empty-classes.uml)  Load patterns (EmptyClass.eiq)  Initialize viewers |
| Várt eredmény | A különböző gráf nézetek vizsgálata egymás után sorban. Az egyes kinézet beállítások után vizsgálat, hogy megtörtént-e a gráf alakzatváltozása. |

|  |  |
| --- | --- |
| Teszteset neve | Testing framework - Add to an existing .eiqsnapshot |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries + Tests Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Lekérdezés eredmények meglévő .eiqsnapshot fájlhoz való adása, és annak vizsgálata, hogy ez megtörtént-e. |

|  |  |
| --- | --- |
| Teszteset neve | Testing framework - Create new .eiqsnapshot |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries + Tests Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school)  Load patterns (simpleSchoolQueries.eiq) |
| Várt eredmény | Új .eiqsnapshot fájl létrehozása, és annak a vizsgálata, hogy a létrehozott példánymodellben megtalálhatóak-e a találatok. |

|  |  |
| --- | --- |
| Teszteset neve | Testing framework - EIQSnapshot Editor - Add new child |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries + Tests Workspace |
| Inicializáló szkriptek | Clear Query Explorer |
| Várt eredmény | EIQSnapshot Editor meglétének ellenőrzése, és a példánymodell módosítása, új gyermek adása a példánymodell egyik eleméhez és ennek a végrehajtásának a vizsgálata. |

|  |  |
| --- | --- |
| Teszteset neve | Validation framework |
| Workbench | IncQuery Workbench |
| Workspace | IncQuery School Queries Workspace |
| Inicializáló szkriptek | Clear Query Explorer  Load instance model (BUTE.school) |
| Várt eredmény | Validation framework inicializálása és a .validation projekt létrehozásának vizsgálata. |

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)
3. Az EMF-IncQuery-t futtató és tesztelő folytonos integrációs környezet [↑](#footnote-ref-3)
4. A Sonar egy folytonos integrációs környezetben használt minőség menedzsment program, amivel analizálni és mérni lehet különböző mennyiségeket adott programra vonatkozóan. [↑](#footnote-ref-4)
5. M, mint Missed, azaz a nem fedett kódrészlet [↑](#footnote-ref-5)
6. C, mint Covered, az a fedett kódrészlet [↑](#footnote-ref-6)