!

SZAKDOLGOZAT-FELADAT

**Stasz Balázs (BD8P7P)**

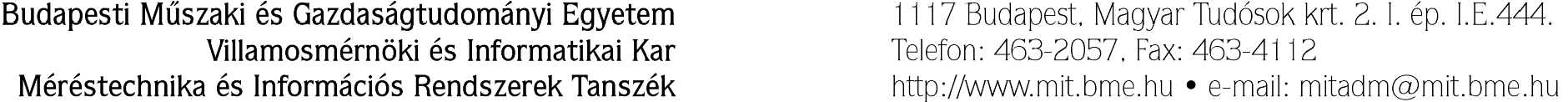
szigorló informatikus hallgató részére

# Felhasználói felületek modellvezérelt tervezése

Napjainkban a modellvezérelt módszerek fontos és hatékony segítséget nyújtanak számos szoftverfejlesztési területen, például a webes és mobil alkalmazások készítése során. A magasszintű modellezési nyelvekkel lehetőség nyílik arra, hogy az alkalmazás számos aspektusát (felhasználói űrlapok, üzleti logika bizonyos részei, kommunikáció) absztrakt mérnöki modellek segítségével tervezzük meg, melyből platformspecifikus modellek, majd a végső kód automatikus transzformációk és kódgenerálás során állnak elő. Ilyen eszközökkel a kézi módszereknél lényegesen hatékonyabban készíthetők komplex szoftverrendszerek, a hatékonyság növekedése fejlesztési költségeket takarít meg (mind a gyorsabb munkavégzés, mind pedig a hibák számának csökkentése, illetve a drága tervezési hibák korai javítása miatt).

A BlackBelt Kft. belső fejlesztési projekt keretében kifejlesztette a Judo keretrendszert, melynek fő célja webes és mobil alkalmazások modellvezérelt fejlesztésének támogatása. A folyamat során a rendszerelemzők által elkészített magas szintű UML modellekből előbb platformfüggetlen, majd platformspecifikus modellek készülnek modelltranszformációk segítségével, a lánc végén a kimenet pedig az alkalmazások prototípusának (vázának) forráskódja is előállítható kódgenerátorok segítségével.

A hallgató feladata, hogy esettanulmány jelleggel kifejlesszen egy a Judo keretrendszerbe beépülő, Eclipse Modeling Framework, valamint Epsilon és VIATRA technológiákra épülő nyílt forrású eszközt, melynek segítségével web platformon futó alkalmazások felhasználói felületének generatív fejlesztése támogatható.



A szakdolgozat kidolgozása a következő részfeladatok megoldását igényli:

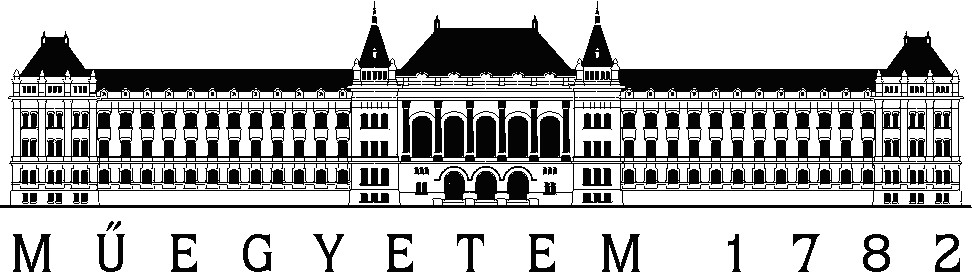
* Végezzen irodalomkutatást, és mutassa be a Judo keretrendszert, valamint a legfontosabb kapcsolódó modellvezérelt technológiákat.
* Tervezzen és valósítson meg egy modellezési nyelvet, mellyel webes alkalmazások felhasználói felülete megtervezhető.
* Tervezzen és valósítson meg automatizált modelltranszformációkat, melyekkel a magasszintű specifikációkat leíró modellekből előállíthatók az alkalmazás-specifikus modellek. Valósítsa meg a modelltranszformációkat inkrementális formában is, melyek lehetővé teszik a későbbi, magas szinten elvégzett modellezési változtatások összefésülését az alacsonyabb szintű modellek változásaival.
* Egy, a VIATRA és az Epsilon modelltranszformációs keretrendszerekben megvalósított transzformációkat összehasonlító esettanulmányon keresztül mutassa be a rendszer működését.

**Tanszéki konzulens:** Dr. Ráth István, adjunktus

**Külső konzulens:** Turi Gyula, szoftvermérnök, BlackBelt Kft.

Budapest, 2017. október 3.

…………………… Dr. Dabóczi Tamás tanszékvezető



Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Felhasználói felületek modellvezérelt tervezése

Szakdolgozat

|  |  |
| --- | --- |
| *Készítette* | *Konzulens* |
| Stasz Balázs | Dr. Ráth István |
|  |  |

2018. december 07.

# Tartalomjegyzék

1. Bevezető
   1. A feladat motivációja
   2. A feladat értelmezése
   3. A dokumentáció tartalma
2. Technikai háttérismeretek
   1. Modelvezérelt szoftverfejlesztés előnyei
   2. Eclipse Modeling Framework (EMF)
   3. Epsilon
   4. VIATRA
   5. JUDO keretrendszer
3. A probléma elemzése
   1. PSM metamodell bemutatása
   2. UI metamodell bemutatása
   3. Traceability metamodell bemutatása
4. A feladat megoldása Epsilon és VIATRA transzformációval
   1. Epsilon transzformáció
      1. Batch transzformáció
      2. ETL fájlok bemutatása
      3. EGL fájlok bemutatása
   2. VIATRA tanszformáció
      1. Eseményvezérelt transzformáció
      2. Rule-ok és Pattern-nek bemutatása
5. A megoldások kiértékelése
   1. Epsilon és VIATRA tanszformációk eredményeinek összehasonlítása
   2. Felhasználói felület leíró értékelése
   3. Batch és Eseményvezérelt transzformációk összehasonlítása
6. Az Epsilon és VIATRA transzformációk használata
   1. Részletes leírás az Epsilon transzformáció lefuttatásához
   2. Részletes leírás a VIATRA transzformáció lefuttatásához
   3. Részletes leírás az összehasonlító teszt futtatásához
7. Összefoglalás

# Bevezetés

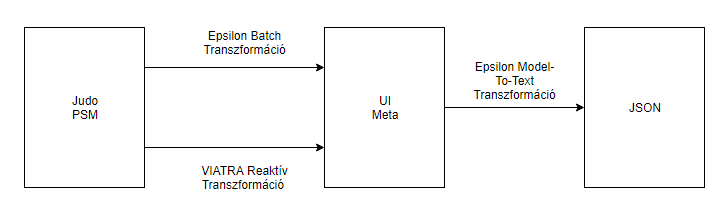
* 1. *A feladatom motivációja*

A Blackbelt Technology Kft belsős projektjeként indult, amelynek én is részese vagyok. Célja a Modelvezérelt szoftverfejlesztési folyamat megreformálása. Eddig a fejlesztési folyamat Modelio és Python alapokon működött, melynek kezelhetőségi, bővíthetőségi és kompatibilitási problémái voltak. Az új folyamat MagicDraw alapokon nyugszik, így felmerült az igény az egész transzformálási folyamat megújítására, amely könnyebben karbantartható és már rendelkezésre áll számtalan bővíthetőségi lehetőség. Ezen kritériumok miatt esett a választás az Eclipse alapú Epsilon és VIATRA keretrendszerekre.

A VIATRA keretrendszer elsősorban a reaktív transzformáció támogatása miatt került előtérbe. A reaktív transzformációban rejlő lehetőségek megkönnyítenék mind az üzleti elemzők, mind a fejlesztői csapat munkáját és felgyorsítaná a model változtatással járó kellemetlenségek megoldását.

* 1. *A feladat értelmezése*

A kitűzött cél egy olyan model transzformációs folyamat felépítése, amely végén egy, a JUDO keretrendszer által feldolgozható JSON formátumú szöveges állomány jön létre, melyből egy alap funkciókra (CRUD) képes felhasználói felület generálható.



A transzformáció kiindulópontja a Blackbelt Kft-től kapott JUDO PSM metamodel, amely egy EMF ECore fájlban van tárolva. Az ezen alapuló modellek transzformációja a következő lépés, amely kétféleképpen történhet. Az első esetben az Epsilonnal megvalósított transzformációval, amely a teljes kiinduló modelt transzformálja, a változtatások számától függetlenül. A második esetben VIATRA reaktív transzformáció alkalmazásával, amely csak az előző állapot óta történt változásokra reagálva végzi el a szükséges transzformációs lépéseket. A transzformáció az általam elkészített UI metamodellre történik, amely egy felhasználói felület általános leírását teszi lehetővé. Végezetül az Epsilonnal megvalósított Model-To-Text transzformáció eredményeképp egy JSON formátumú állomány jön létre, amely a JUDO keretrendszernek megfelelő formázással és tartalommal rendelkezik, így előállítható belőle egy működő felhasználói felület.

A szakdolgozat terjedelme a JUDO PSM modellből való transzformációtól indul és a JSON állomány előállításáig tart, beleértve a UI metamodell előzetes elkészítését is.

*1.3 A dokumentáció tartalma*

A második fejezetben bemutatásra kerülnek a modellvezérelt szoftverfejlesztés előnyei, illetve a felhasznált technológiák rövid ismertetése. Ezek ismerete elengedhetetlen az elkövetkezendő fejezetek megértéséhez.

A harmadik fejezetben a két, transzformációban használt metamodell részletes leírása olvasható.

A negyedik fejezet két fő részből áll. Először az Epsilon transzformációhoz létrehozott ETL fájlok részletes bemutatása történik, ezt követi a kód generáláshoz létrehozott EGL fájlok bemutatása. A második részben a VIATRA transzformációhoz felhasznált Rule és Pattern fájlok leírása olvasható. A fejezet igyekszik választ adni a transzformációs szabályok mögött megbúvó logikára.

Az ötödik fejezetben a transzformációk és generátorok eredményeinek kiértékelése és összehasonlítása olvasható. Ezen felül egy példa UI, amely az Epsilon által előállított JSON-ből keletkezett a JUDO keretrendszer felhasználásával.

A hatodik fejezetben három leírás található, amelyek tartalmazzák az Epsilon és VIATRA transzformációk futtatásának módját, illetve ezek eredményének összehasonlítására szolgáló teszt futtatását.

Végezetül, a hetedik fejezetben egy rövid összefoglalás olvasható a feladat elvégzése közben szerzett tapasztalatokról.

# Technikai háttérismeretek

* 1. *Modelvezérelt szoftverfejlesztés előnyei*

A modellvezérelt szoftverfejlesztés előtérbe kerülése több tényezőnek köszönhető. Ezen tényezők mind a fejlesztési folyamatra, mind az üzleti kiadásokra pozitív hatással vannak.

Az első ilyen tényező az automatizált kódgenerálás, amely használatával lehetőség van gyakran használt kódrészletek automatikus generálására. Például egy osztály esetén a kötelező létrehozás/törlés/frissítés (CRUD) operációk. Ezzel jelentős idő és pénz takarítható meg. A fejlesztői csapatnak elegendő az általános funkcionalitáson túlmutató logikát implementálnia.

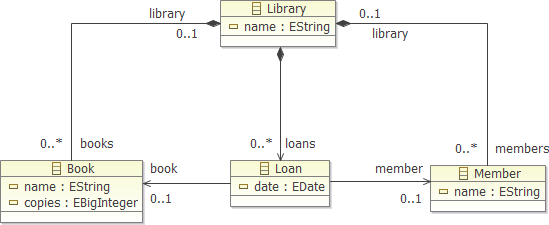
A transzformáció megkezdése előtt lehetőség van validálni a modelt. Így még a modell transzformáció megkezdése előtt kiderül, ha tervezési időben hiba csúszott a kivitelezésbe.

Segíti a szoftver dokumentálását. Egy jól megtervezett modell már önmagában megkönnyíti az összefüggések megértését, kiemeli az összetartozásokat.

Mivel a platform-független (PIM) modell nem vesz tudomást arról, hogy milyen architektúrán fogják a készülő szoftvert implementálni, így elegendő a szemtantikai működést megtervezni és a késöbbiekben, ha a szükség úgy hozza, architekturális változások a modell változtatása nélkül végrehajthatóak.

*2.2 Eclipse Modeling Framework (EMF)*

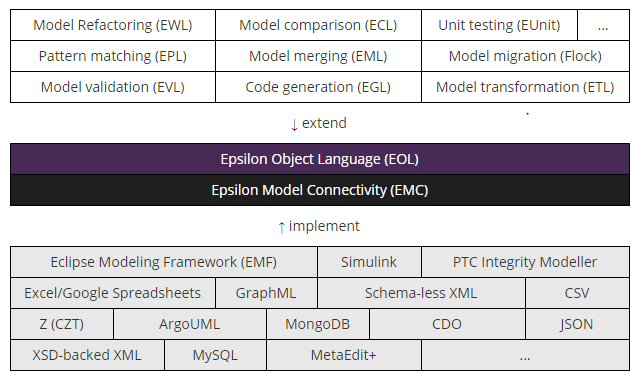
Az Eclipse Modeling Framework (EMF) egy modellező és kódgeneráló keretrendszer. Az ECore metamodell az EMF alapeleme, melynek segítségével kifejezhető bármely más modell, akár önmaga is. Az ECore metamodelleken alapuló modelleket hozhatunk létre, melyek az ECore metamodelljükben megadott eszközkészlettel rendelkeznek.



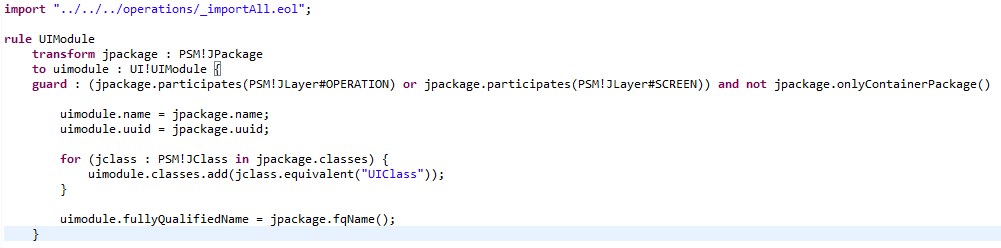
Az alábbi példán egy könyvtár egyszerű metamodellje látható. Egy könyvtár (Library) objektumnak lehetnek könyvei (Book), tagjai (Member), kikölcsönzései (Loan), illetve neve (name), amely String típusú.

*2.3 Epsilon*

Az Epsilon nyelvek és eszközök gyűteménye, amelyeket Model-To-Model transzformációra, Model-To-Text transzformációra, modell validálásra, modell összehasonlításra terveztek.

Az Epsilon magja az Epsilon Object Language (EOL), egy imperatív programozási nyelv, amely EMF modellek létrehozására, módosítására és lekérdezésére készült.

*Epsilon Transformation Language (ETL)*: Szabály alapú Model-To-Model transzformációs nyelv, amely támogatja a több modellből több modellbe való transzformálást, szabályok öröklését, a lazy és greedy annotációkat, és mind a bemeneti, mind a kimeneti modellek módosítását és lekérdezését.



Az *import* kulcsszóval lehet a használni kívánt metódusokat elérhetővé tenni a szabályban. Például az onlyContainerPackage() és a fqName().

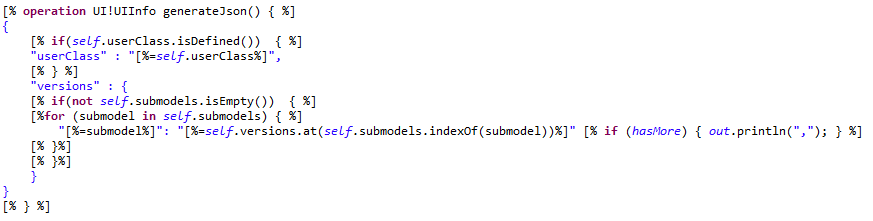
A *rule* kulcsszó után kell megadni a szabály nevét, ez jelen esetben UIModule. A *transform* kulcsszó után három adatot kell megadni. Az első a transzformálandó változó neve, ahogy hivatkozni akarunk rá a szabályban, ez teljesen tetszőleges. A második, hogy melyik modellből származik az elem, a példában ez a PSM. A harmadik, hogy milyen típusú a transzformálandó elem, a példában JPackage.

A *to* kulcsszó után kell megadni, hogy milyen elem lesz a transzformáció eredménye. Szintaxisa megegyszik a *transform* szintaxisával*.*

A *guard* segítségével lehet megszűrni a transzformálandó modell elemeket. Csak azokra az elemekre fut meg a szabály, amelyekre teljesül a *guard*-ban definiált feltétel.

Az *equivalent* az Epsilon saját metódusa, amely ETL transzformációk futtatására alkalmas. Paraméterként átadható a futtatandó szabály neve, vagy paraméter nélkül meghívva az Epsilon választja ki az általa megfelelőnek ítélt szabályt. Visszatérési értéke a transzformált objektum, vagy null, ha nem sikerült a transzformáció. Nyomon követi a transzformált objektumokat, így nincs szükség külső Traceability modellre.

*Epsilon Generation Language (EGL)*: Sablon alapú Model-To-Text nyelv kód és dokumentáció generáláshoz.



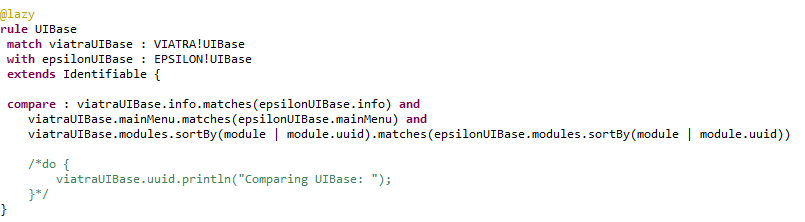
Az alábbi EGL metódus generálja egy UIInfo JUDO kompatibilis JSON leíróját.

Az *operation* kulcsszó után adható meg, hogy milyen objektumon hívható meg a metódus, a példában UI!UIInfo. A metóduson belül a hivatkozott objektumot a *self* kulcsszóval lehet elérni.

A *[% %]* szimbólumokkal jelzett, úgynevezett dinamikus részek nem kerülnek automatikusan a kimenetre. Ezek alkalmasak elágazások, ciklusok kezelésére. A *[%=kif%]* a *[%out.print(kif)%]* egyszerűsített változata.

Minden, ami nem *[% %]* szimbólumokkal határolt, úgynevezett statikus rész, automatikusan a kimenetre kerül.

*Epsilon Compare Language (ECL)*: Szabály alapú nyelv, amely alkalmas homogén vagy heterogén metamodelleken alapuló modellek összehasonlítására.



A *@lazy* annotációval jelölhető, hogy egy szabály ne fusson meg automatikusan, csak ha kódból meg van hívva.

A *rule* kulcsszó után adható meg a szabály neve, ahogy hivatkozni szeretnénk rá a kódban.

A *match* és *with* kulcsszavak után adható meg, hogy melyik modellből milyen elemet akarunk összehasonlítani.

Az *extends* kulcsszó után adható meg annak a szabálynak a neve, amelyből származtatni szeretnénk.

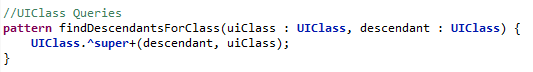
A *compare* kulcsszó után adhatóak meg a feltételek, melyek teljesülése esetén a két modell elem egyezőnek tekinthető. A *matches* metódus az Epsilon beépített metódusa, meghívja az Epsilon által megfelelőnek ítélt ECL szabályt.

A *do* kulcsszó után írhatóak azok az utasítások, melyeket egy sikeres összehasonlítás után le akarunk futtatni.

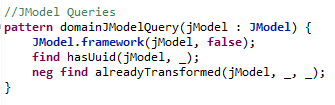
*2.4 VIATRA*

A VIATRA egy modell és adat transzformációs keretrendszer, amely esemény vezérelt és reaktív transzformációkra fókuszál.

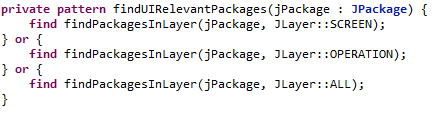
*VIATRA Query Language* (VQL) : Gráf mintaillesztésre alapuló nyelv, amellyel a modell elemeket lehet a kívánt megkötések és feltételek mellett lekérdezni. A megkötéseket, illetve feltételeket a minta (*pattern*) törzsében kell megfogalmazni. Eredménye a paraméterekből képzett találatok halmaza, ahol minden találat csak egyszer szerepelhet. A paramétereknek meg lehet adni az típusát.



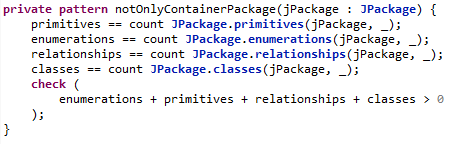
Az alábbi minta (*pattern*) egy UIClass példányhoz – descendant – megkeresi az összes ősosztályt – uiClass. A super+ jelentése tranzitív lezárt, azaz a super asszociáció mentén az összes descendant-ból kiinduló ősosztályt visszadja. Minden descendant – uiClass páros csak egyszer szerepelhet az eredmény halmazban.



Az alábbi minta (*pattern*) a feltételeknek megfelelő JModel példányok halmazával tér vissza. A *JModel.framework(jModel, false)* megkötés kiszűri azon JModel elemeket, amelyek *framework* attribútuma nem *false* értékű. A *find* kulcsszóval lehet más mintákat (*pattern*) meghívni. A \_ jelöléssel olyan értékek adhatóak meg, amelyek értékére nem vagyunk kíváncsiak, csak a létezésükre. Tehát a *find hasUuid(jModel, \_)* kiszűri azon JModel elemeket, amelyeknek a UUID attribútuma nincs megadva. A *neg* kulcsszóval ellátott megkötések akkor igazak, ha az adott paraméterekkel egy helyes találat sincs. A *neg find alreadyTransformed(jModel, \_, \_)*  megkötés teljesül, ha a jModel elem még nem lett transzformálva.



Az *or* kulcsszóval lehet diszjunkciót kifejezni. Egy találat helyes, ha bármelyik törzsre nézve igaz.

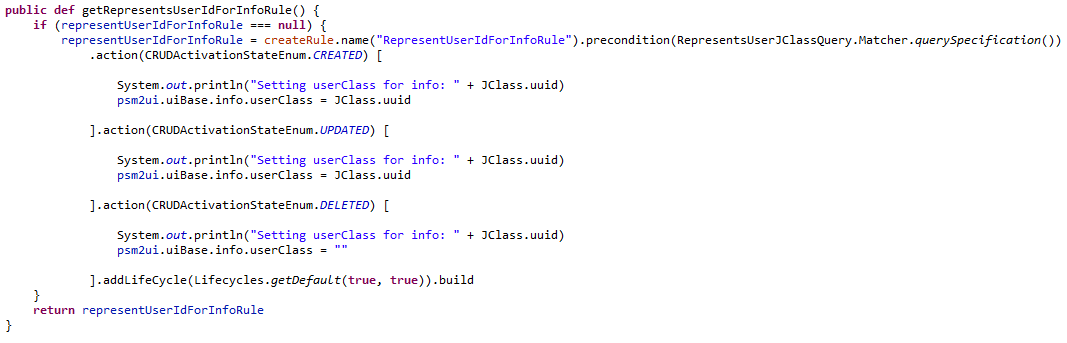


A *check* blokk ad lehetőséget arra, hogy attribútumokon érték vizsgálatot lehessen végrehajtani. A vizsgálat feltétele, hogy mellékhatás mentesnek kell lennie.

*Transzformációs szabályok*: A szabályok Xtend nyelven lettek implementálva.

Egy transzformációs szabály vezérlését minden esetben egy minta (*pattern*) végzi. A minta eredmény halmazának változása okozza az egyes szabályok ágainak lefutását.

* Ha a mintának az eredményhalmaza kibővül egy új találattal, akkor a szabály Create ága fut le az új találat paramétereivel.
* Ha egy találat megszűnik, akkor a Delete ág fut le a megszűnt találat paramétereivel.
* Ha egy találatban valamely elem attribútuma változik meg, de maga a találat nem módosul, akkor a szabály Update ága fut meg.



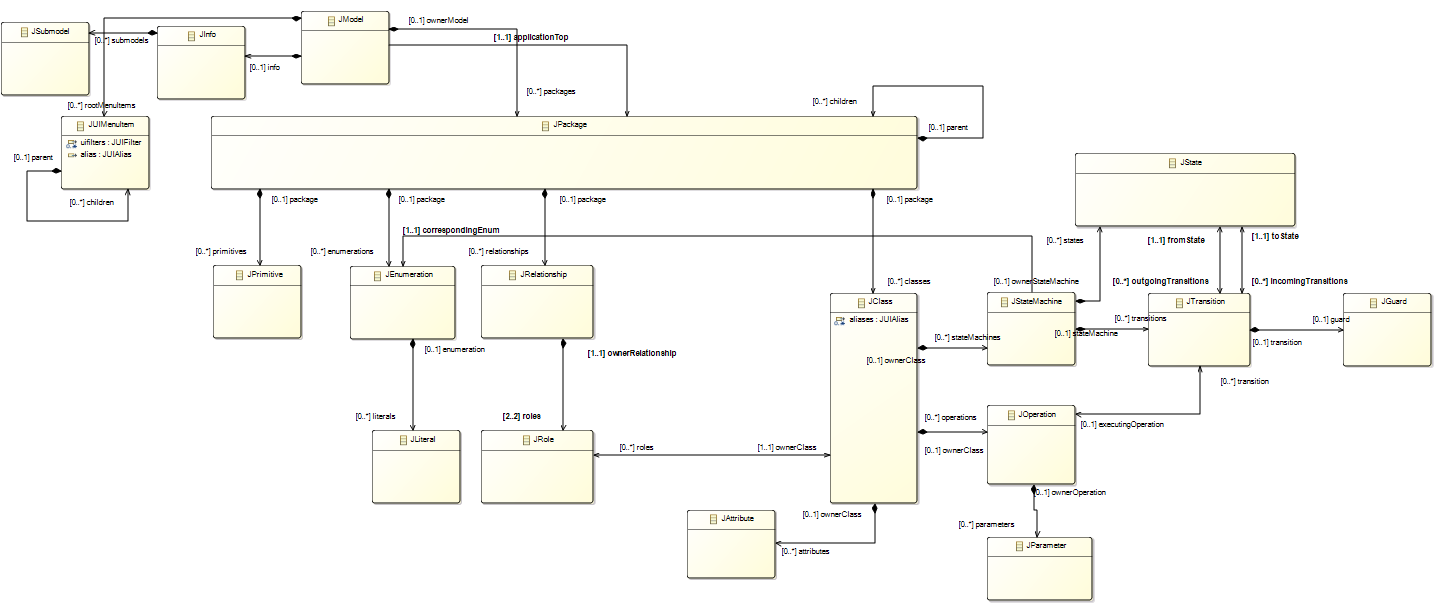
* 1. *JUDO keretrendszer*

A JUDO keretrendszer a Blackbelt Technology Kft. saját fejlesztésű Model-To-Model transzformációs keretrendszere. A JUDO keretrendszer működése többszörös Model-To-Model és Model-To-Text transzformációkon alapszik. Az üzleti elemzők által megtervezett UML modellből először egy JUDO specifikus PSM modell készül. Ebből a PSM modellből transzformálódik például a UI, Java és RDBMS modellek. Végezetül pedig ezen modellek Model-To-Text transzformációjával keletkeznek azon állományok, melyeket a JUDO keretrendszer feldolgoz és önmagában működő szoftvert állít elő belőle. Az alap funkciók a CRUD műveleteket foglalják magukba.

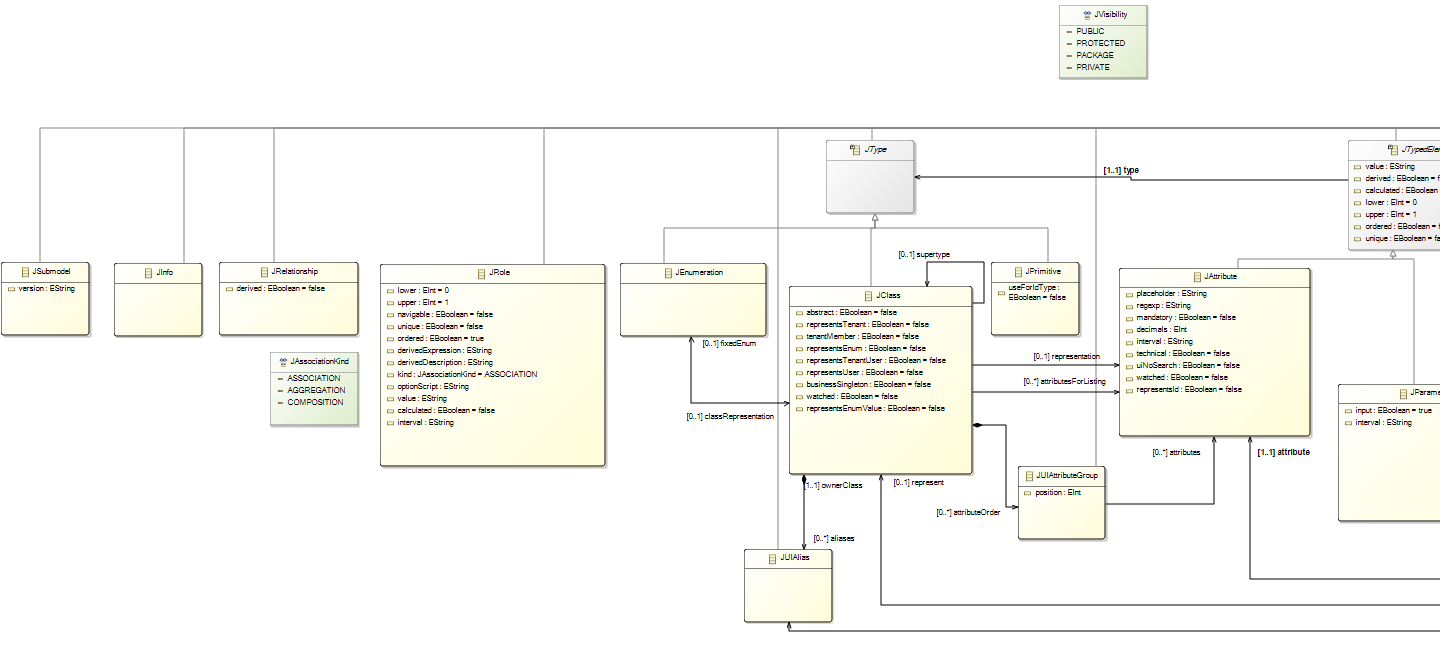
# A metamodellek elemzése

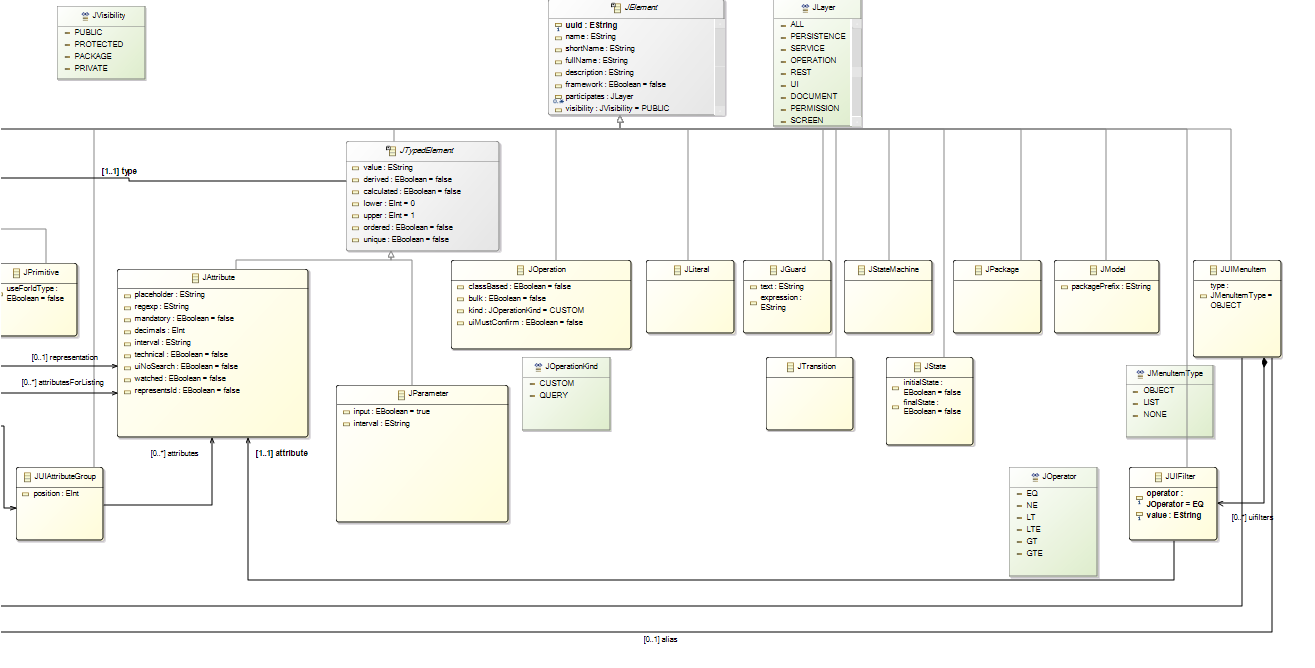
* 1. *PSM metamodell bemutatása*

PSM metamodell psm\_domain nézete:



PSM metamodell psm\_types nézete:



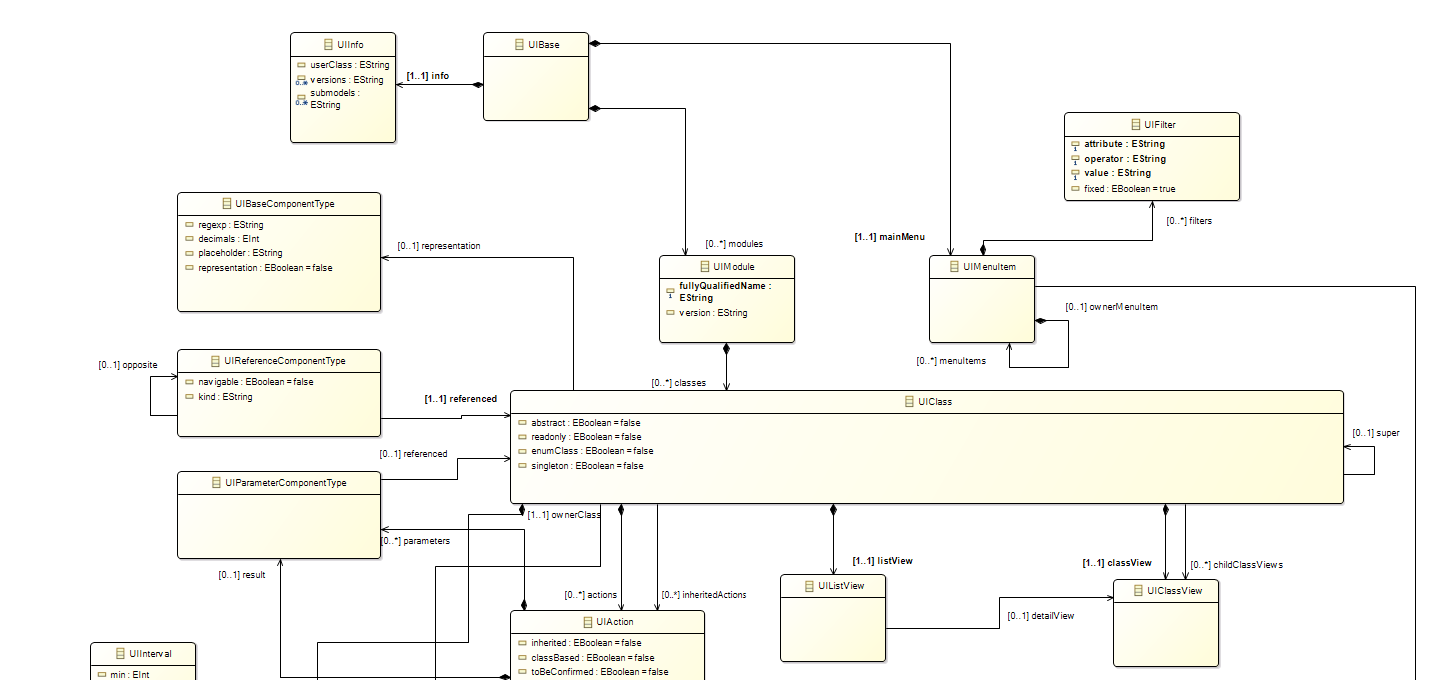


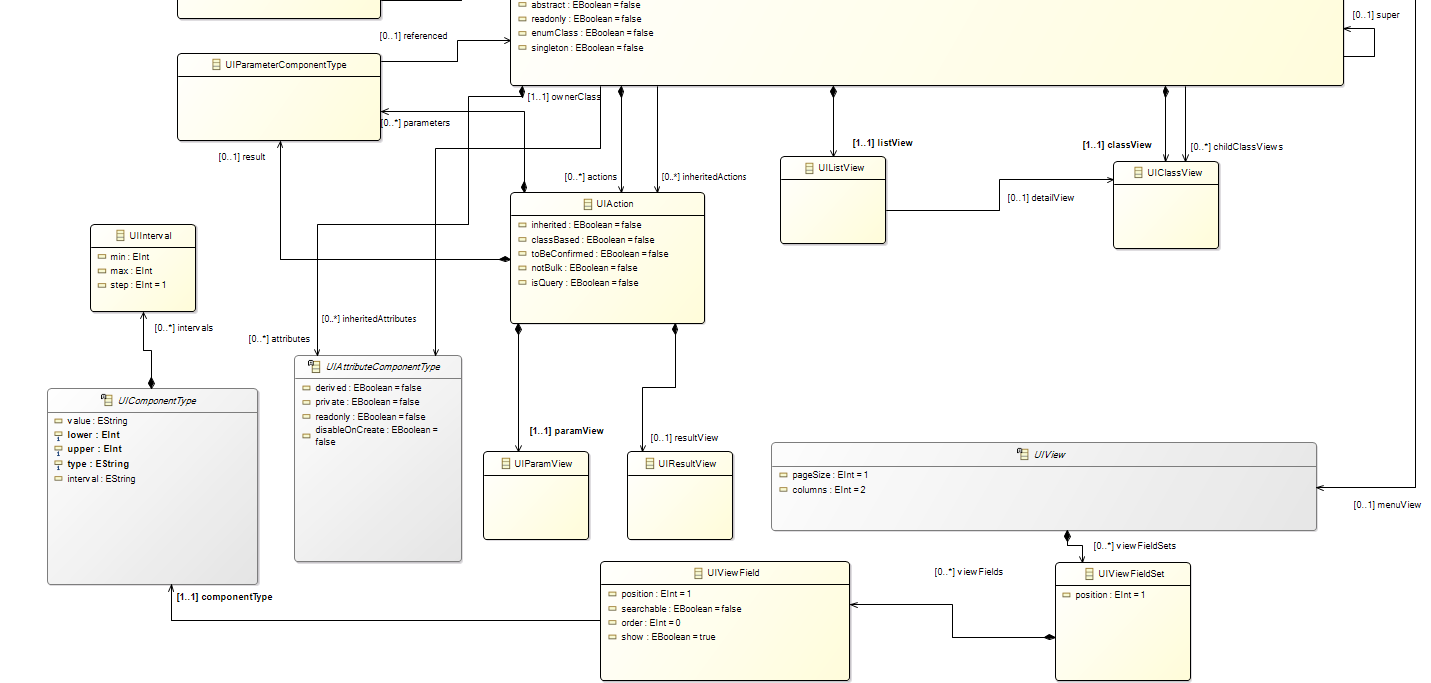
A PSM metamodell átmeneti modellként szolgál az UML és a domain specifikus modellek között, így elkészítésénél a fő szempont volt, hogy tartalmazzon mindent, amire a UI, Java és RDBMS metamodelleknek szüksége lehet.

Elkészítésénél csak kisegítő munkát végeztem, célja, hogy kiinduló pontként szolgáljon a transzformációhoz.

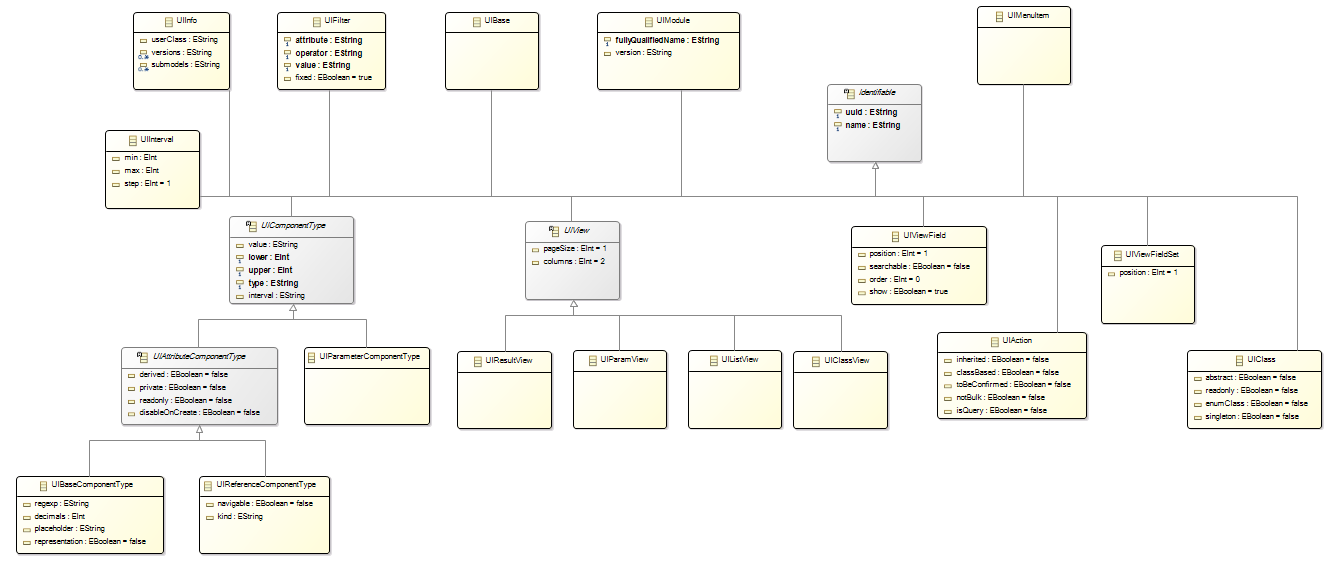
* 1. *UI metamodell bemutatása*

UI metamodell ui\_domain nézete:





UI metamodell ui\_types nézete:



A UI metamodell tervezésénél a fő szempont volt, hogy a JUDO keretrendszer által ismert JSON struktúra egyszerűen előállítható legyen Model-To-Text transzformációval egy UI modellből.

Identifiable: Absztrakt osztály. Minden UI metamodellben szereplő osztály őse, így biztosítva, hogy minden elemnek kötelező legyen a uuid és name attribútum. Az elemek uuid attribútuma teszi lehetővé a pontos, egyszerű azonosítást.

UIBase: JUIInfo elemből transzformálódik. A UI modell gyökér eleme.

UIInfo: JInfo elemből transzformálódik. A felhasznált modellek (pl PIM UML modell), keretrendszerek (pl. JUDO), modellező eszközök (pl. MagicDraw, Modelio) nevét, verzióját tartalmazza. A userClass attribútuma tartalmazza annak az osztálynak a nevét, amely a felhasználót (User) reprezentálja.

UIMenuItem: JUIMenuItem elemből transzformálódik. A felhasználói felület legnagyobb egysége. Tartalmazhat más menüket, vagy egy adott osztály nézetét jeleníti meg a menuView asszociáción keresztül. ListView vagy ClassView nézetet jeleníthet meg.

UIFilter: JUIFilter elemből transzformálódik. Alapértelmezett szűrőként funkcionál azon a Menü képernyőn, amelyet a hozzá tartozó UIMenuItem meghatároz. A szűrő feltételeit az objektum attribútumai határozzák meg.

UIModule: JPackage elemből transzformálódik. Funkciója a UIClass elemek rendszerezése.

UIClass: JClass elemből transzformálódik. Az UML modellben definiált osztályokat reprezentálja. A hozzá tartozó ListView és ClassView kezeléséhez tartalmaz információkat, illetve UIReferenceComponentType, UIParameterComponentType típusaként használható.

UIListView: Egy UIClass elemhez definiált nézet. Az adott UIClass típusú objektumokat jeleníti meg lista nézetben. A ListView képernyőkről lehet tovább navigálni a ClassView képernyőkre egy objektum kiválasztásával. Például: Autó osztály esetén kilistázza az Autó objektumokat, és egy konkrétat kiválasztva átnavigál az adott objektum ClassView nézetére.

UIClassView: Egy UIClass elemhez definiált nézet. Az adott UIClass típusú objektumot részletező nézetben jelenít meg. Itt tekinthető meg az összes attribútuma, hívhatóak meg rajta a metódusai.

UIAction: JOperation elemből transzformálódik. Az UML modellben definiált metódusokat reprezentálja.

UIParamView: Egy UIAction elemhez definiált nézet. A UIAction alapértelmezett nézete, ez jelenik meg metódus meghívásakor a felületen, ahol a paraméterek értékei egyesével megadhatóak.

UIResultView: Egy UIAction elemhez definiált nézet. A UIAction visszatérési értékét jeleníti meg a képernyőn.

UIViewFieldSet: Segítségével a UIViewField elemek csoportokba szervezhetőek a UIView elemeken.

UIViewField: A UIBaseComponentType, UIReferenceComponentType és UIParameterComponentType típusú objektumok megjelenítéséért felelős.

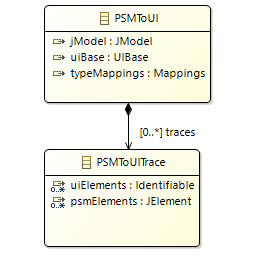
UIBaseComponentType: JAttribute elemből transzformálódik. Az UML modellben definiált alap típusú (pl. Integer, String) attribútumokat reprezentálja.

UIReferenceComponentType: JRole elemből transzformálódik. Az UML modellben definiált asszociációkat (pl Autó - Kerék) reprezentálja. Implementációs okokból az UML modellben Enum típusú attribútumként definiált elemből is JRole transzformálódik, így UIReferenceComponentType lesz belőle.

UIParameterComponentType: JParameter elemből transzformálódik. Az UML modellben metódus paramétereként definiált elemeket reprezentálja.

UIInterval: JUIInterval elemből transzformálódik. A UIBaseComponentType és az alap típusú UIParameterComponentType elemek értékét szabályozza. Beállítható minimum és maximum érték, illetve a két határérték közötti értékek távolsága.

* 1. *Traceability metamodell bemutatása*



A Traceability metamodell szolgál a transzformációban résztvevő modell elemek nyomonkövetésére. A Traceability metamodell, és az ez alapján létrehozott modell csak a VIATRA transzformáció során kerül felhasználásra, az Epsilon transzformációnak saját, belső nyomkövetési módszere van.

PSMToUI: A Traceability metamodell központi eleme, összeköti a PSM modell JModel (jModel) és a UI modell UIBase (uiBase) elemét. Ezen kívül még tartalmazza a PSM alaptípusainak (pl: String, Number) leképzését UI alaptípusokká (typeMappings), melyek a psmToUiTypeMapping modellben találhatóak meg.

PSMToUITrace: Az egyes PSM modell elemekhez társítja a belőlük keletkezett UI modell elemet. Segítségével nyomonkövethető, hogy mely elemek lettek már transzformálva. Bár a metamodell szerint egy Trace-hez több PSM, illetve UI elem tartozhat, a jelenlegi transzformáció során 1-1 kapcsolat áll csak fent.

# A feladat megoldása Epsilon és VIATRA transzformációval

* 1. *Epsilon transzformáció*
     1. *Batch transzformáció*

Batch transzformáció futtatásakor a kiinduló modell egészét transzformáljuk, függetlenül attól, hogy előző transzformáció óta történt-e benne változás.

* + 1. *ETL fájlok bemutatása*

A transzformációs szabályok ETL (Epsilon Transformation Language) fájlokban vannak leírva, a szabályokhoz esetlegesen szükséges metódusok pedig EOL (Epsilon Object Language) fájlokban lettek definiálva.

uibase.etl: JModel elemből transzformál UIBase elemet.

*Lefutás feltétele*: Ha még nincs UIBase és a JModel framework attribútuma false.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* A PSM modellben lévő összes JPackage elemre meghívja a transzformációs szabályt és a UIBase alá rendeli őket.
* Létrehozza a fő UIMenuItem elemet, amelyet a main\_menu uuid-val lát el.
* A PSM modellben lévő összes JUIMenuItem elemre meghívja a transzformációs szabályt, és a main\_menu alá rendeli őket.
* Létrehozza a UIInfo eleme és beállítja a szükséges attribútumokat.

uimodule.etl: JPackage elemből transzformál UIModule elembe.

*Lefutás feltétele*: A JPackage elemnek Screen vagy Operation besorolással kell rendelkeznie és nem lehet csak konténer, azaz tartalmaznia kell JPackage elemtől különböző elemet.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* A JPackage JClass elemeire meghívja a transzformációs szabályt és a UIModule alá rendeli őket.

uimenuitem.etl: JUIMenuItem elemből transzformál UIMenuItem elemet.

*Lefutás feltétele*: A JUIMenuItem elemnek a parent asszociációja értelmezve legyen.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* A JUIMenuItem JUIMenuItem elemeire meghívja a transzformációs szabályt és a UIMenuItem alá rendeli őket.

uifilter.etl: JUIFilter elemből transzformál UIFilter elemet.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.

uiaction.etl: JOperation elemből transzformál UIAction elemet.

*Lefutás feltétele*: A JOperation láthatósága (visibility) legyen PUBLIC.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* Végigmegy a JOperation paraméterein és meghívja a transzformációs szabályt.
  + Ha bemeneti a paraméter, hozzáadja a UIAction paramétereihez.
  + Ha kimeneti a paraméter, beállítja a UIAction kimeneti paramétereként és létrehozza a kapcsolódó ResultView, ViewFieldSet és ViewField objektumokat.
* Létrehozza a kapcsolódó ParamView és ViewFieldSet objektumokat, majd minden paraméterhez létrehoz egy ViewField objektumot és hozzárendeli a ViewFieldSet-hez.

uiclass.etl: JClass elemből transzformál UIClass elemet.

*Lefutás feltétele*: A JClass elemet tartalmazó JPackage elemnek Screen vagy Operation besorolással kell rendelkeznie.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* A JClass elemet reprezentáló attribútumot transzformálja és beállítja a UIClass reprezentációjának. A reprezentáló attribútum értéke jelenik meg, ha a UIClass megjelenik például ListView nézetben a képernyőn.
* A JClass elem ősét transzformálja és beállítja a UIClass ősét.
* A JClass JAttribute/JRole/JOperation elemeit transzformálja és beállítja a UIClass megfelelő asszociációjához.

uicomponenttype.etl: A JRole, JAttribute és JParameter transzformálási szabályait tartalmazza.

*UIReferenceComponentType rule:* JRole elemből transzformál UIReferenceComponentType elemet.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* A JRole ellenkező oldalát (opposite) jelentő JRole elemet transzformálja és beállítja a UIReferenceComponentType megfelelő asszociációját.
* A JRole típusául szolgáló JClass elemet transzformálja és beállítja a UIReferenceComponentType típusának.
* Létrehozza a szükséges UIInterval elemeket.

*UIBaseComponentType rule*: JAttribute elemből transzformál UIBaseComponentType elemet.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* Létrehozza a szükséges UIInterval elemeket.

*UIParameterComponentType rule*: JParameter elemből transzformál UIParameterComponentType elemet.

*Lefutás feltétele*: A JParameter elemet tartalmazó JOperation láthatósága legyen PUBLIC.

*Rule lefutása*:

* Beállítja a szükséges attribútumokat.
* Ha a JParameter típusa JClass, akkor transzformálja a JClass elemet és beállítja a UIParameterComponentType típusának. Ha a JParameter típusa alaptípus, kikeresi a UI modell szerinti alaptípust és azt állítja be típusnak.
* Létrehozza a szükséges UIInterval elemeket.

psmToUi.etl: A transzformáció belépési pontja. Az itt importált ETL fájlokban található szabályok futnak meg a transzformáció során. Lehetőség van *pre,* illetve *post* transzformációs fázist létrehozni. A *pre* fázis a transzformációs szabályok előtt fut meg, a *post* pedig a szabályok lefutása után.

A *pre* fázis jelen transzformációban nem került felhasználásra.

*post*:

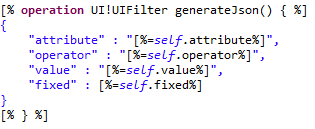
* Beállítja a UIClass elemek örökölt attribútumait és metódusait.
  + A teljes leszármazási fából megkapják az elemeket.
  + Eredetileg a UIClass transzformációja során került beállításra, de nem a kívánt eredményt adta vissza.
* Létrehozza a UIClass elemekhez a ListView és ClassView elemeket.
  + A ListView létrehozása során egy ViewFieldSet jön létre, ez alá kerülnek be az attribútumok és öröklött attribútumok ViewField elemei.
  + Az ObjectView esetén először a JClass JAttributeGroup elemei által hivatkozott attribútumok ViewField elemei jönnek létre, melyek a JAttributeGroup alapján létrehozott ViewFieldSet alá kerülnek. A maradék attribútumok saját ViewFieldSet és ViewField elemet kapnak.
* Beállítja a UIMenuItem elemeken, hogy melyik ListView vagy ObjectView elemet hivatkozzák.
  + 1. *EGL fájlok bemutatása*

A felhasználó felület előállításához szükséges JSON állomány generálását végző kód kettő EGL fájlban került implementálásra.

main.egl: Ez az EGL futtatás belépési pontja. Lefutása két részből tevődik össze. Először megfuttatja a jsonMain.egl fájlt, majd a futtatás eredményét a ui-meta.json fájlba írja.

jsonMain.egl: Összeállítja a sablon tartalmát, egyesével végigmenve a modell elemeken. A kiindulási pontja a UIBase elem, innen az asszociációkon keresztül járja be a modell elemeit. Az egyes elemek reprezentációt a generateJson() metódus állítja elő, amely minden típusra meg lett írva.

Például a UIFilter modell elemek reprezentációját generáló generateJson() metódus:



* 1. *VIATRA transzformáció*
     1. *Eseményvezérelt transzformáció*

Eseményvezérelt transzformáció futtatásakor nincs szükség a teljes kiindulási modellt transzformálni, csak az előző transzformáció óta történt változásokat, vagy a transzformáció során történő módosításokat.

* + 1. *Rule-ok és Pattern-nek bemutatása*

Az összes Pattern a PatternProvider.vql fájlban található, kommentel jelölve, hogy melyik modell elemhez tartoznak. A transzformációs szabályok modell elemek szerint kategorizálva külön fájlokba kerültek.

EventDrivenPsmToUi.xtend: A transzformáció kiindulási pontja. Létrehozza a transzformációt, hozzárendeli a transzformációs szabályokat, beállítja a prioritásukat, végezetül pedig elindítja a lefutást.

RuleProvider.xtend: Az EventDrivenPsmToUi használja transzformációs szabályok lekérdezésére. Rajta keresztül érhetőek el a szabályok Getter metódusokkal.

ModelRuleFactory.xtend:

*DomainJModelQuery minta*:

* Paraméterei: JModel jModel.
* Az eredménye azon JModel példányok, amelyek még nem lettek transzformálva, a UUID attribútumuk be van állítva és nem a keretrendszer részei (jModel.framework = false). Elvárt működés esetén egy ilyen JModel elemnek szabad csak szerepelnie a modellben.

*ModelRule szabály*:

* Minta: DomainJModelQuery
* Aktivációs állapotok: Create
* Beállítja a UI modell UIBase elemének attribútumait.
* A Traceability modellben létrehozza a Trace elemet a jModel és UIBase kapcsolatához.
* Létrehozza a fő UIMenuItem elemet main\_menu néven és létrehozza a UIInfo elemet main\_info néven. Ezen elemek nem kerülnek a Traceability modellben eltárolásra.

*DomainJModelQueryForModify minta:*

* Paraméterei: JModel jModel
* Eredménye azon JModel példányok, amelyek transzformálva lettek. Elvárt működés esetén egy ilyen JModel elemnek szabad csak szerepelnie a modellben.

*ModifyModelRule szabály:*

* Minta: DomainJModelQueryForModify
* Aktivációs állapotok: Update
* A Domain modellen csak a Name és UUID attribútumok módosítása engedélyezett. A szabály beállítja ezen új értékeket a UIBase modell elemnek. Elvárt használat mellett a domain modell törlése nem engedélyezett, ezért a Delete ág nem került implementálásra.

PackageRuleFactory.xtend:

*JPackageToUIModuleQuery minta:*

* Paraméterei: JPackage jPackage
* Eredménye azon JPackage példányok, amelyek tartalmaznak JPackage elemen kívül más modell elemet is (pl: JClass, JPrimitive), a megfelelő rétegben helyezkednek el (SCREEN vagy OPERATION), a Name és UUID attribútumaik be lettek állítva, illetve még nem lettek transzformálva.

*PackageRule szabály:*

* Minta: JPackageToUIModuleQuery
* Aktivációs állapotok: Create
* Létrehozza a UIModule elemet.
* Beállítja a szükséges attribútumokat.
* Végezetül pedig hozzáad a Traceability modellhez egy JPackage - UIModule Trace elemet.

*JPackageToUIModuleQueryForModify minta:*

* Paraméterei: JPackage jPackage, PSMToUITrace trace
* Eredménye a transzformált JPackage elemek és a hozzájuk tartozó Trace elem.

*ModifyPackageRule szabály:*

* Minta: JPackageToUIModuleQueryForModify
* Aktivációs állapotok: Update, Delete
* *Update*: A frissített jPackage attribútum értékek alapján beállítja a UIModule új értékeit.
* *Delete*: Törli a UIModule elemet a UI modellből, illetve törli a trace elemet a Traceability modellből.

ClassRuleFactory.xtend:

*JClassWithGuardConditionQuery minta:*

* Paraméterei: JClass jClass, UIModule uiModule
* Eredménye azon JClass példányok, amelyek a transzformált JPackage elemekhez tartoznak. A UIModule elemek a JPackage UI megfelelői.

*ClassRule szabály*:

* Minta: JClassWithGuardConditionQuery
* Aktivációs állapotok: Create
* Létrehozza a UIClass példányt és a hozzátartozó Trace elemet, ha még nem léteznek. Amennyiben már igen, a UIModule példányhoz rendeli a UIClass példányt.
* Beállítja a szükséges attribútumokat.
* Ha a JClass elemnek van ősosztálya, megnézi, hogy transzformálva lett-e már. Ha igen, beállítja a transzformálás alatt álló UIClass supertype attribútumát. Ha nem, létrehozza az ősosztálynak megfelelő UIClass elemet, de nem rendeli UIModule elemhez, illetve létrehozza a Trace elemet. Ez az oka a transzformáció elején történő vizsgálatnak.
* Végezetül létrehozza a UIClass példányhoz tartozó UIListView és UIClassView elemeket. Mivel a UIListView elemnek csak egyetlen UIViewFieldSet eleme van, így az is itt jön létre. A UIView, illetve UIViewFieldSet elemekhez nem jön létre Trace elem, mivel nem PSM elemből transzformálódnak.

*JClassWithGuardConditionQueryForModify minta:*

* Paraméterei: JClass jClass, PSMToUITrace trace
* Eredménye azon JClass elemek, amelyek transzformálva lettek, illetve a hozzájuk tartozó Trace elemek.

*ModifyClassRule szabály:*

* Minta: JClassWithGuardConditionQueryForModify
* Aktivációs állapotok: Update, Delete
* *Update*: Lekéri a Trace alapján a UIClass példányt.
* Frissíti az attribútum értékeket a UIClass, UIListView, UIClassView és a UIListView UIViewFieldSet elemeken.
* *Delete*: Törli a UIClass példányt és a hozzá tartozó Trace elemet.

AttributeGroupRuleFactory.xtend:

*JAttributeGroupQuery minta:*

* Paraméterei: JUIAttributeGroup jAttributeGroup, UIClassView classView
* Eredménye azon JUIAttributeGroup példányok, amelyek még nem lettek transzformálva és UUID attribútumuk be van állítva, illetve a JUIAttributeGroup példányokhoz tartozó JClass elemek UIClass megfelelőjükhöz tartozó UIClassView példányok.

*AttributeGroupRule szabály:*

* Minta: JAttributeGroupQuery
* Aktivációs állapotok: Create
* Létrehoz egy UIViewFieldSet példányt, amely attribútumait beállítja a JUIAttributeGroup példány alapján. Végül létrehoz egy Trace elemet.

*JAttributeGroupQueryForModify minta:*

* Paraméterei: JUIAttributeGroup jAttributeGroup, UIClassView classView, PSMToUITrace trace
* Eredménye azon JUIAttributeGroup példányok, amelyek már transzformálva lettek, a JUIAttributeGroup példányokhoz tartozó JClass elemek UIClass megfelelőjükhöz tartozó UIClassView példányok, illetve a JUIAttributeGroup példányokhoz tartozó Trace elemek.

*ModifyAttributeGroupRule szabály:*

* Minta: JAttributeGroupQueryForModify
* Aktivációs állapotok: Update, Delete
* Update: Lekéri a UIViewFieldSet példányt trace segítségével, majd frissíti az attribútumait jAttributeGroup és classView attribútumai alapján.
* Delete: Törli a UIViewFieldSet példányt, illetve a trace-t.

AttributeRuleFactory.xtend:

*JAttributeQuery minta:*

* Paraméterei: JAttribute jAttribute, UIClass uiClass, UIBaseComponentType baseType
* Eredményei azon JAttribute elemek, amelyekhez már be lett állítva a UUID attribútum és még nem lettek transzformálva, illetve a hozzájuk tartozó JPrimitive és JClass példányok UI megfelelői.

*AttributeRule szabály:*

* Minta: JAttributeQuery
* Aktivációs állapotok: Create
* Létrehozza a UIBaseComponentType példányt, majd beállítja az attribútumait jAttribute, uiClass és baseType attribútumai alapján.
* Létrehozza a UIBaseComponentType példányhoz tartozó UIInterval elemeket.
* Létrehozza a Trace elemet jAttribute és a UIBaseComponentType példány alapján.
* Meghívja a createViewFields metódust, amely uiClass classView és listView példányához létrehozza a UIBaseComponentType elemhez tartozó UIViewFieldSet és UIViewField elemeket. A classView elemhez tartozó UIViewFieldSet és UIViewField csak akkor jön létre, ha a uiClass JClass megfelelőjéhez tartozó JUIAttributeGroup elemek egyike sem hivatkozik a jAttribute elemre.
* Lekéri az összes UIClass példányt, amely a uiClass leszármazottja. Az inheretedAttributes listájukhoz hozzáadja a UIBaseComponentType példányt, majd meghívja a createViewFields() metódust az aktuális UIClass példánnyal paraméterként.

*JAttributeQueryForModify minta*

* Paraméterei: JAttribute jAttribute, UIClass uiClass, UIBaseComponentType baseType, PSMToUITrace trace
* Eredményei azon JAttribute elemek, amelyekhez már transzformálva lettek, illetve a hozzájuk tartozó trace és JPrimitive, JClass példányok UI megfelelői.

*ModifyAttributeRule szabály:*

* Minta: JAttributeQueryForModify
* Aktivációs állapotok: Update, Delete
* Update: Lekéri a jAttribute UI megfelelőjét trace segítségével, majd frissíti az attribútumait új értékekkel.
* Delete: Törli a jAttribute UI megfelelőjét a UI modellből, a trace elemet a Trace modellből.

*JAttributeInJGroupViewFieldQuery minta:*

* Paraméterei: JAttribute jAttribute, UIViewFieldSet viewFieldSet, UIBaseComponentType uiBaseType, UIClassView classView
* Eredménye azon JAttribute elemek, amelyek valamelyik JUIAttributeGroup példányhoz tartoznak. A viewFieldSet a JUIAttributeGroup UI modellbeli megfelelője, a uiBaseType a jAttribute UI modell beli megfelelője, a classView pedig a viewFieldSet elemet tartalmazó példány.

*AttributeInGroupViewFieldRule szabály:*

* Minta: JAttributeInJGroupViewFieldQuery
* Aktivációs állapotok: Create
* A viewFieldSet példányhoz létrehoz egy új UIViewField elemet és beállítja az attribútumait.

*JAttributeViewFieldQueryForModify minta:*

* Paraméterei: JAttribute jAttribute, UIClass uiClass, UIBaseComponentType uiBaseType

*ModifyAttributeViewFieldRule szabály:*

*FindCorrespondingType minta:*

*PrimitiveRule szabály:*

RoleRuleFactory.xtend:

*JRoleQuery minta:*

*RoleRule szabály:*

*JRoleQueryForModify minta:*

*ModifyRoleRule szabály:*

*JRoleViewFieldQuery minta:*

*RoleViewFieldRule szabály:*

OperationRuleFactory.xtend:

*JOperationWithGuardQuery minta:*

*OperationRule szabály:*

*JOperationWithGuardQueryForModify minta:*

*ModifyOperationRule szabály:*

ParameterRuleFactory.xtend:

*JParameterWithGuardQuery minta:*

*ParameterRule szabály:*

*JParameterWithGuardQueryForModify minta:*

*ModifyParameterRule szabály:*

*JParameterViewFieldQuery minta:*

*ParameterViewFieldRule szabály:*

MenuRuleFactory.xtend:

*FindMenuItemWithParent minta:*

*MenuRule szabály:*

*FindMenuItemToSetParent minta:*

*MenuParentSetterRule szabály:*

*FindMenuItemWithParentForModify minta:*

*ModifyMenuRule szabály:*

FilterRuleFactory.xtend:

*JUIFilterQuery minta:*

*FilterRule szabály:*

*JUIFilterQueryForModify minta:*

*ModifyFilterRule szabály:*

InfoRuleFactory.xtend:

*JInfoQuery minta:*

* Paraméterei: JInfo jInfo
* Eredménye azon JInfo elemek, amelyek UUID attribútuma meg van adva és még nem lettek transzformálva. Elvárt működés esetén maximálisan egy ilyen van.

*InfoRule szabály:*

* Minta: JInfoQuery
* Aktivációs állapot: Create
* A JModel transzformációja során létrehozott main\_info UIInfo elemen beállítja a versions és submodels attribútumokat. Létrehoz egy Trace elemet, amellyel összekapcsolja a jInfo és main\_info példányokat.

*RepresentsUserJClassQuery minta:*

*RepresentsUserIdForInfoRule szabály:*

*JInfoQueryForModify minta:*

*ModifyInfoRule szabály:*

# A megoldások kiértékelése

*5.1 Epsilon és VIATRA transzformációk eredményeinek összehasonlítása*

Az összehasonlítás *Epsilon Compare Language (ECL)* használatával történik. A két összehasonlítandó modell az Epsilon transzformáció által generált modell és a VIATRA transzformáció által újonnan generált modell, az inkrementális szabályok megfuttatása nélkül.

Az összehasonlító algoritmus két fájlban lett implementálva.

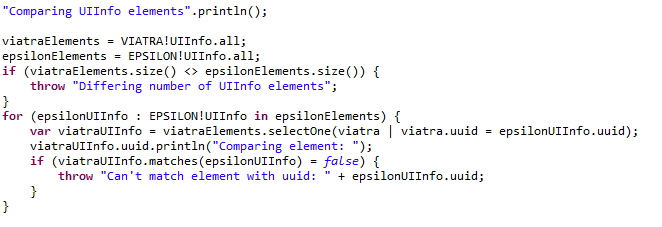
modelComparisonTest.ecl:

Itt vannak implementálva az összehasonlító szabályok. Minden modell elem típusért külön szabály felelős. Az összehasonlítás sikeres, ha két elem attribútumainak az értékei megegyeznek, ha a referenciák ugyan arra az elemre mutatnak, illetve, ha lista esetén a lista tartalma megegyezik a sorrend figyelmen kívül hagyásával. A szabályok el lettek látva a @lazy annotációval, így csak akkor futnak meg, amikor explicite meg lettek hívva. Ennek elhagyása jelentős teljesítménybeli romláshoz vezet. Egyes referenciák összehasonlításakor csak a UUID attribútumok van összehasonlítva. Ez a limitáció a körök feloldása miatt volt szükséges.



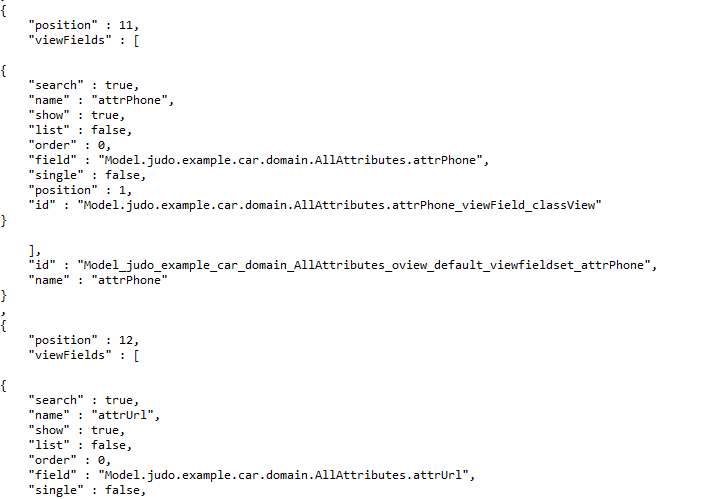
mainTest.ecl:

Összehasonlítja a modell elemeket típusonként. Először ellenőrzi, hogy egy adott típusból megegyező számú van-e a két modellben. Ha nem, „Differing number of ... elements” hibaüzenettel megszakad a lefutás. Ha igen, végigiterál az Epsilon modellen. Minden elemhez UUID alapján megpróbálja megtalálni a VIATRA elemet, és meghívja rá a matches() beépített eljárás segítségével a modelComparisonTest-ben található megfelelő összehasonlító szabályt. Ha a szabály nem megfelelőnek találja a két elemet, „Can’t match element with uuid: ...” hibaüzenettel véget ér a futás.



*5.2 Felhasználói felület leíró értékelése*

A leíró JSON formátumban generálódik. A létrejövő állomány szintaktikailag helyes, de formázatlan, így emberi olvasásra nem igazán alkalmas. A leíró célja, hogy a JUDO keretrendszer számára értelmezhető struktúrában szolgáltasson felület leírást, így jön létre belőle egy már működő felhasználó felület. A JSON struktúrája fa szerűen épül fel, ahol az egyes elemek a modell elemekből nyert információk segítségével kerülnek előállításra.



*5.3 Batch és Eseményvezérelt transzformációk összehasonlítása*

Mind a Batch, mind az Eseményvezérelt tanszformációnak megvannak az előnyei és a hátrányai.

A Batch transzformációt számottevően egyszerűbb implementálni, mint Eseményvezérelt változatát. A Create, Update és Delete esetekből csak a Create esetén lefutó szabályokat kell megvalósítani. Batch transzformáció futtatásakor a generált modell biztosan üres, így nem kell felkészíteni a kódot arra, hogy lekezelje az esetlegesen már létező elemeket. Mivel Eseményvezérelt esetben csak a megváltozott állapotnak megfelelő szabályok futnak meg, így az új elemeket be kell illeszteni, a törölt elemeket pedig eltávolítani.

Az egyszerűbb implementációból fakadóan mind időben, mind pénzügyi szempontból jelentős megtakarítást jelenthet csak a Batch transzformáció elkészítése, de hosszú távon veszteséges is lehet. Minden transzformáláskor a teljes modellt újra kell transzformálni, még akkor is, ha csak néhány új elem változott meg. Ez modelltől és a transzformációs szabályoktól függően több órát is igénybe vehet, amely hátráltathatja a fejlesztést és szignifikáns anyagi veszteséget jelenthet.

Választás előtt érdemes átgondolni, hogy az Eseményvezérelt transzformáció implementálásához szükséges plusz erőforrások megtérülnek-e a használat során nyert előnyökkel.

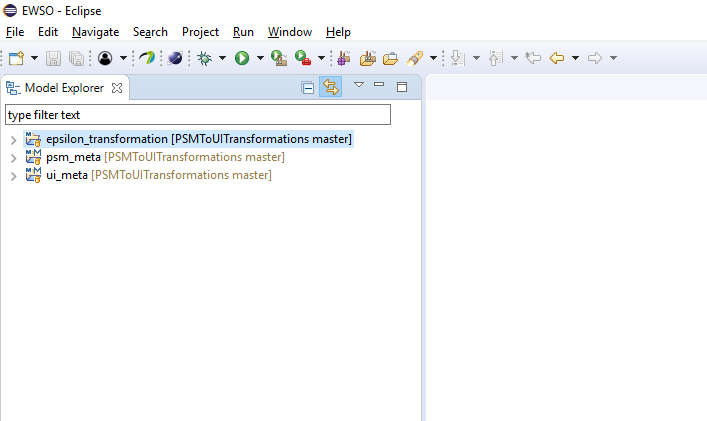
# Az Epsilon és VIATRA transzformációk használata

A transzformációk Eclipse Oxygen 3.A IDE használatával készültek (<https://www.eclipse.org/oxygen/>), Oxygen előtti verzió nem támogatott.

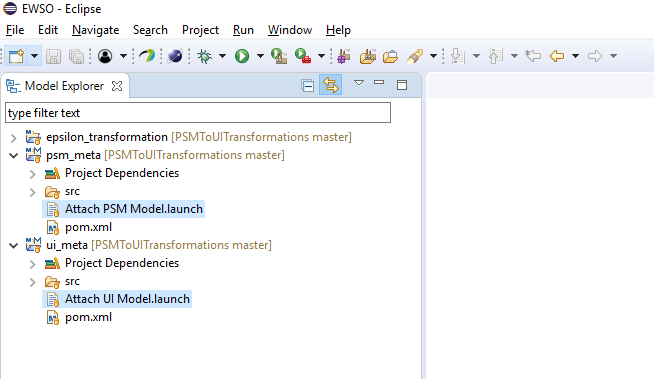
*6.1 Részletes leírás az Epsilon transzformáció lefuttatásához*

Az Epsilon transzformációk futtatásához Maven plugin, illetve Epsilon plugin telepítése szükséges.

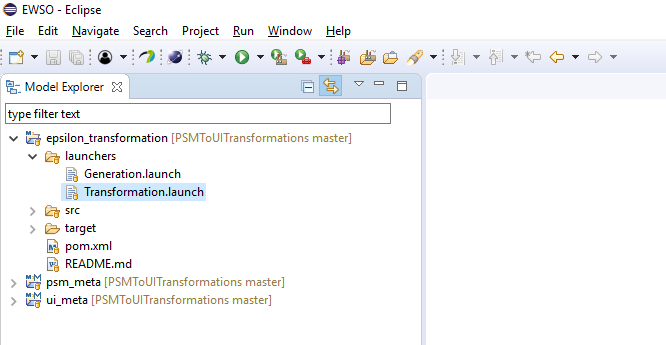
A *PSMToUITransformations* projektben lévő *Epsilon\_Transformation* projekt tartalmát kell betölteni Eclipse workspace-be.



A *psm\_meta* és *ui\_meta* projektekben található egy *Attach PSM Model.launch*, illetve *Attach UI Model.launch* fájl, ezeket futtatva adhatóak hozzá a transzformációhoz szükséges metamodellek és modellek.



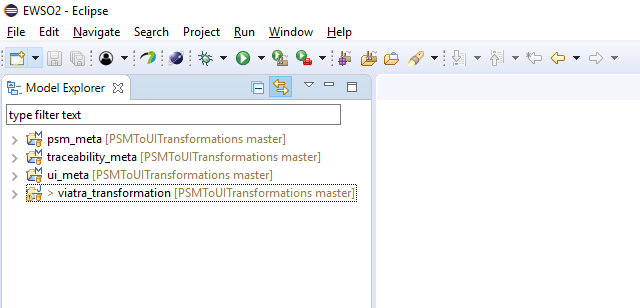
Az *epsilon\_transformation* projekt *launchers* mappájában található *Transformation.launch* futtatásával hajtható végre a Model-To-Model transzformáció. A transzformált modellt a *target* mappába rakja *ui.model* néven. A *Generation.launch* a Model-To-Text transzformációt futtatja meg, a generált JSON állományt *ui-meta.json* néven szintén a *target* mappába rakja.



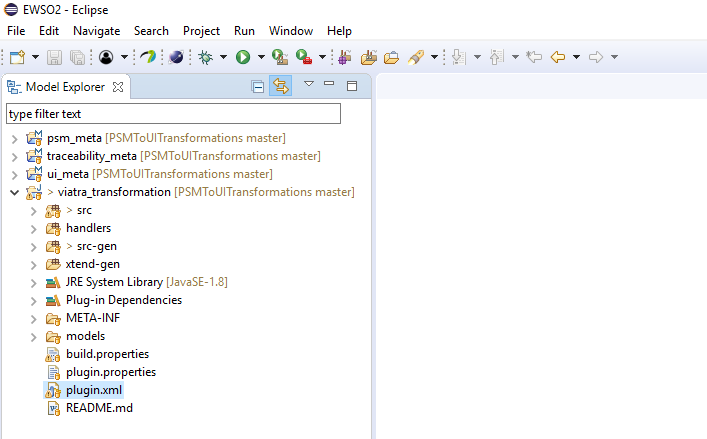
*6.2 Részletes leírás a VIATRA transzformáció lefuttatásához*

A VIATRA transzformáció futtatásához VIATRA 2.0.2+ plugin telepítése szükséges.

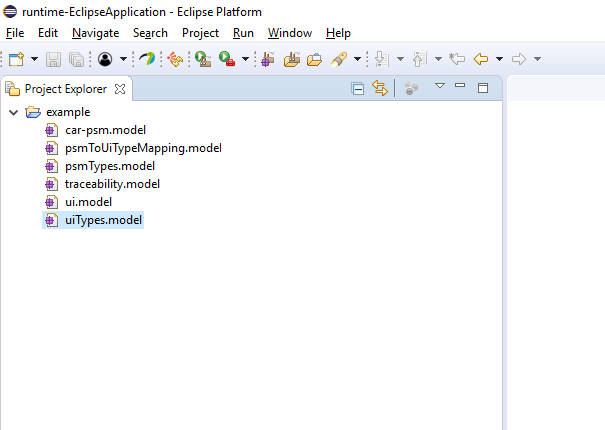
A *PSMToUITransformations* projektben lévő *VIATRA\_Transformation* projekt tartalmát kell betölteni Eclipse workspace-be.



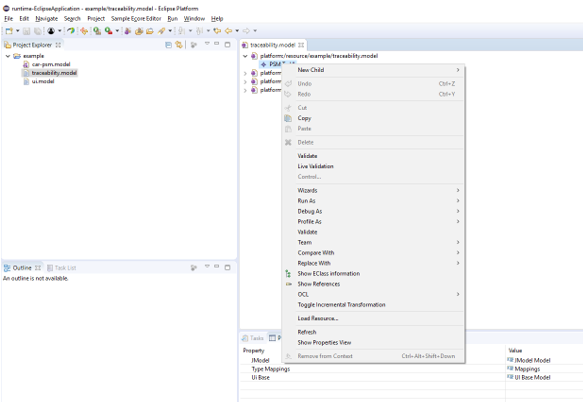
A *viatra\_transformation* projektet futtatva elindul a runtime Eclipse-t.



A runtime Eclipse elindulása után egy generikus projektet kell létrehozni, melybe a *viatra\_transformation/models* mappájában lévő modelleket kell bemásolni.



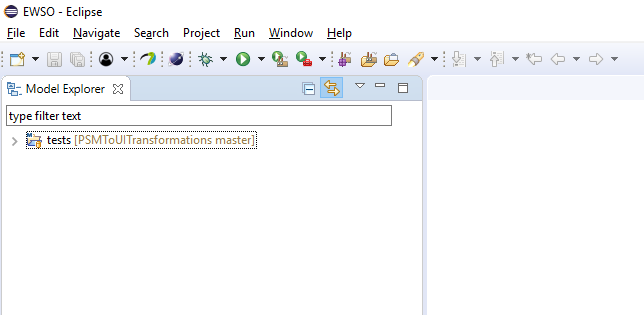
A *traceability.model*-t *Sample Ecore Model Editor*-ral megnyitva lehetőségünk van elindítani a transzformációt. A *traceability.model* PSMToUI elemén jobb klikk, utána *Toggle Incremental Transformation* aktiválása. A kezdeti elemek transzformálásának befejeztével, amely az indítás után automatikusan megtörténik, a PSM modellt szerkesztve aktiválhatjuk az inkrementális transzformációkat. Fontos, hogy a PSM modellt ugyan abban az editorban szerkesszük, mint amiben a Traceability modellt megnyitottuk.



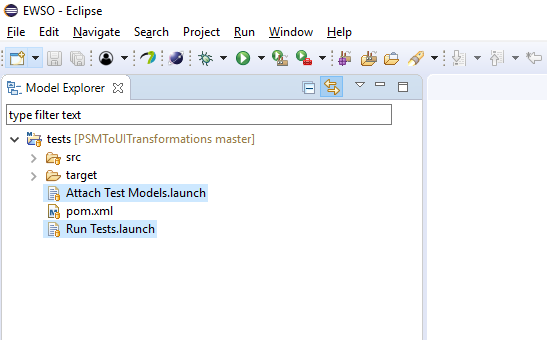
*6.3 Részletes leírás az összehasonlító teszt futtatásához*

A teszt futtatásához Maven plugin, illetve Epsilon plugin telepítése szükséges. Ezeken felül az Epsilon transzformációhoz is kellő UI metamodellek és modellek be kell legyenek töltve (Attach UI Models.launch).

A *PSMToUITransformations* projektben lévő *Tests* projekt tartalmát kell betölteni Eclipse workspace-be.



Az *Attach Test Models.launch* futtatásával lehet az összehasonlítandó modelleket betölteni. A modellek a *src/models* mappában találhatóak. A *Run Tests.launch* futtatásával lefut a teszt, amely összehasonlítja a két transzformációval készült modellt.



# Összefoglalás

//TODO

# Hivatkozások

A modellvezérelt szoftverfejlesztés előnyei:

<http://www.theenterprisearchitect.eu/blog/2009/11/25/15-reasons-why-you-should-start-using-model-driven-development/>

Epsilon dokumentáció:

<https://www.eclipse.org/epsilon/doc/>

VIATRA dokumentáció:

<https://www.eclipse.org/viatra/documentation/tutorial.html>

Eclipse Oxygen:

<https://www.eclipse.org/oxygen/>