VIATRA Query Language

A VIATRA legújabb kiadás verziója a 2.7.0. Korábban az EMF-IncQuery-nek hívták.

A VIATRA alapértelmezetten elérhető az Eclipse Modeling Tools distribúcióban.

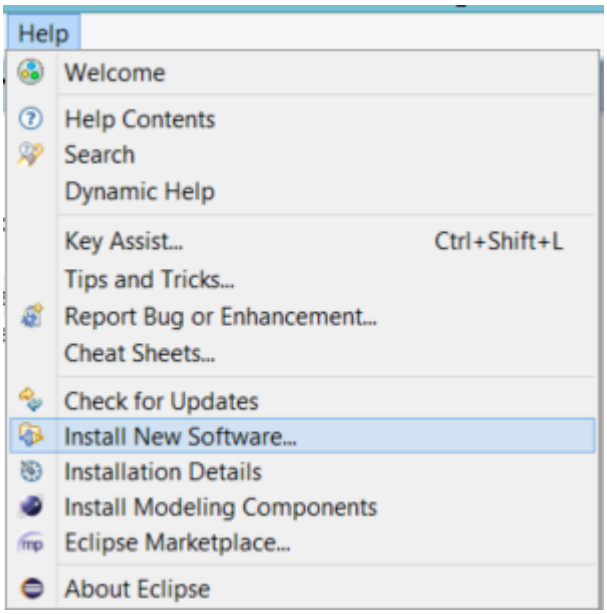
honlap: https://www.eclipse.org/viatra/

VIATRA Query telepítése

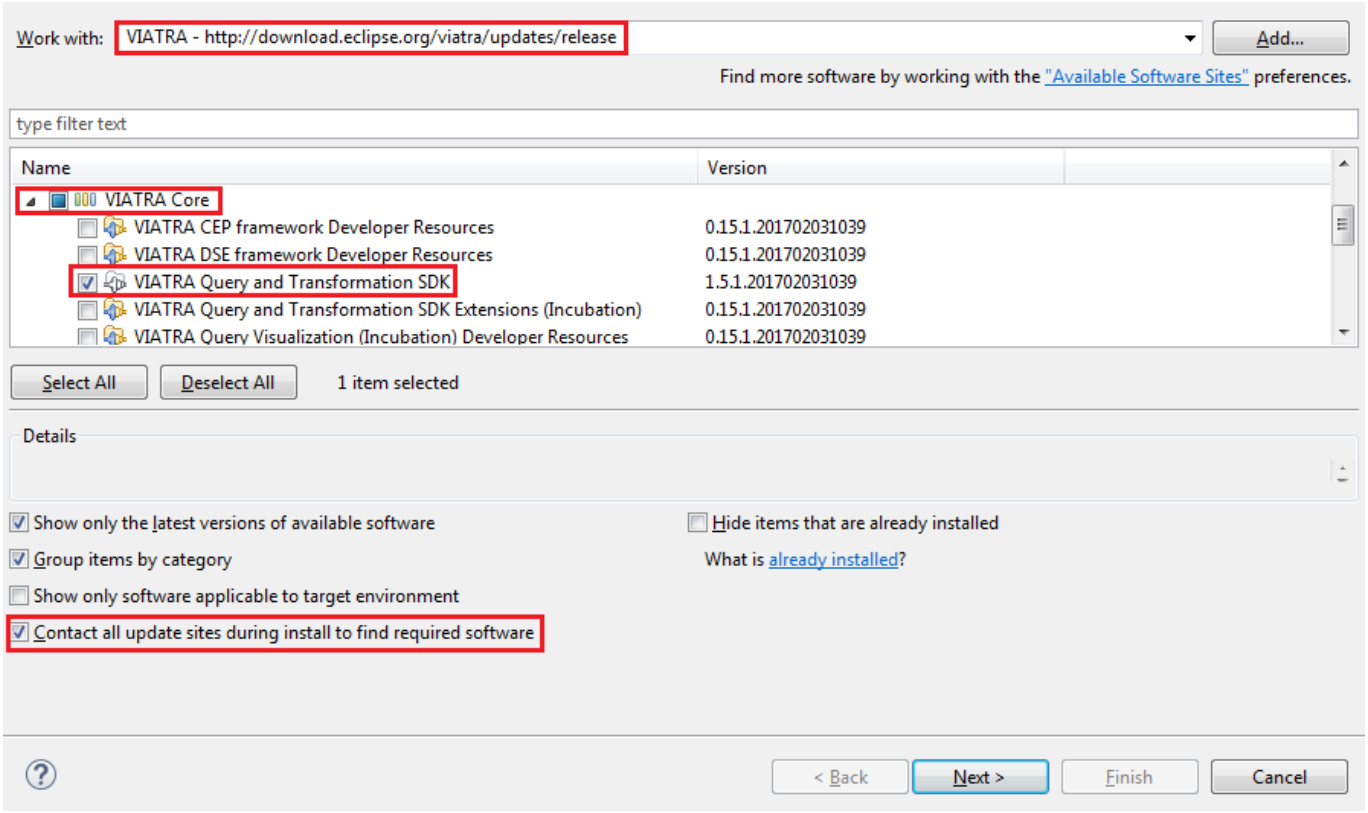
Navigáljon a VIATRA honlapjára és keressen "update site"-okat a letöltési oldalon:
https://eclipse.org/viatra/downloads.php

Keresse meg a VIATRA *release* update site-ot, majd másolja a vágólapra (ne hagyjon white space-t):
http://download.eclipse.org/viatra/updates/release/latest

Váltson vissza az Eclipse-re, majd válassza a *Help/Install New Software...* lehetőséget.



Illessze be a kimásolt URL-t a *Work with* mezőbe, majd nyomja meg az *Enter* gombot. Amikor a nézet frissül, válassza ki a *VIATRA Query and Transformation SDK* lehetőséget. Jelölje be a *Contact all update sites during install...* mezőt. Nyomja meg a *Next*, majd a *Next*, majd a *Finish* gombot. A telepítési folyamat után újra kell indítania az Eclipse-et.



Azoknak a haladó felhasználóknak, akik gyorsabb telepítést szeretnének, ajánlott az *Contact all update sites during install...* mező kijelölése nélkül telepíteni, de lehet, hogy az *Xtend* és *Xtext* technológiákat manuálisan kell telepíteni.

Megjegyzés az forráskód példákhoz

Az alábbi példák közül több végállapota is elérhető a https://github.com/FTSRG/mdsd-examples repository-ban, a VQL branch-en; egészen pontosan a vql_2017 és a vql_2020 tag-ek alatt.

Projektek előkészítése

1. Klónozza és importálja az alábbi projekteket a git repository-ból: https://github.com/FTSRG/mdsd-examples

   ```
   hu.bme.mit.mdsd.erdiagram
   hu.bme.mit.mdsd.erdiagram.examplediagrams
   hu.bme.mit.mdsd.erdiagram.example
   ```

2. Hozzon létre egy új *Query Project*-et, és nevezze el `hu.bme.mit.mdsd.erdiagram.queries`-nek.

3. Adja hozzá a függőséget a `hu.bme.mit.mdsd.erdiagram` projektben a manifest fájl használatával.

4. Hozzon létre egy új query definíciót a `hu.bme.mit.mdsd.erdiagram.queries` package-ben, egy `queries.vql` nevű fájlban (megjegyzés: általában érdemes leíróbb nevet választani). Továbbá adjon hozzá két egyszerű lekérdezést is (ne felejtse el menteni és buildelni):

   ```
   ```

```
package hu.bme.mit.mdsd.erdiagram.queries

// Az alábbi kódrészlet importálja az ecore modellt,
// az auto-completion használatához üssünk a ctrl+space billentyűkombinációt
az idézőjel után
import "hu.bme.mit.mdsd.erdiagram"

pattern entity(e : Entity) {
    Entity(e);
}

pattern entityName(entity : Entity, name) {
    Entity.name(entity, name);
}
```

Ha nem adta hozzá a függőséget az erdiagram projekthez, akkor hibaüzenet jelenik meg az import parancsnál. Elérhető egy quick fix a függőség hozzáadására.

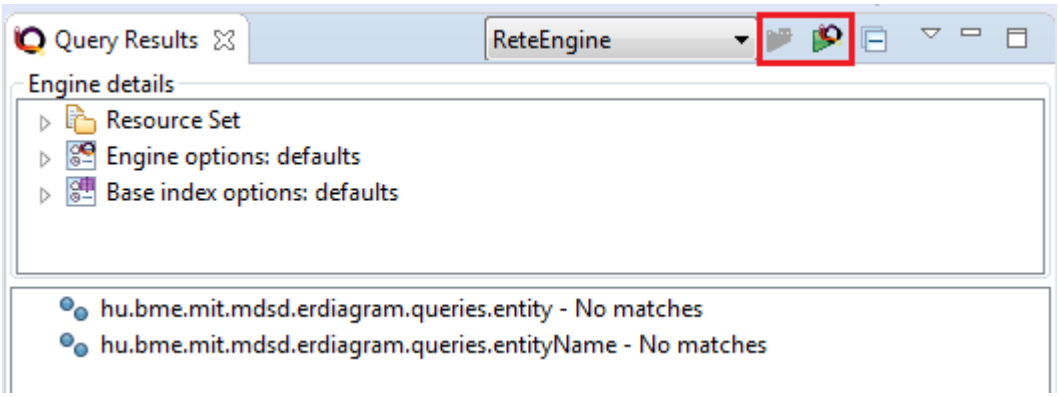
Ahogy látható, minden mintának van egyedi neve és több paramétere. Minden paraméternek van egy típusa, amit el lehet hagyni, de érdemes explicit megadni. A minták testén belül különböző *kényszerek* vannak. Az első példánk típuskényszert ír le, a második pedig feature kényszert. Az első azt állítja, hogy az `entity` változó `Entity` eClass típusú, a második pedig, hogy az `entity` változó `name` attribútuma a `name` változó értékével egyezik meg.

Query Results nézet

A **Query Results** (Lekérdezés eredmények) az elsődleges eszköz a VIATRA lekérdezések debugolásához a fejlesztés során. A nézet megnyitásához: *Window/Show View/Others -> VIATRA/Query Results* vagy egyszerűen nyomja meg a *CTRL + 3* billentyűkombinációt, majd kezdje el írni a nézet nevét.

Ennek az ablaknak a használatához be kell töltenünk egy példány modellt és egy query készletet az ablakba:

1. Nyissa meg a példa példánymodellt (*My.erdiagram*) a Sample Reflective Ecore Model szerkesztővel.
2. Kattintson az ablakon található zöld nyílra a példánymodell betöltéséhez az ablakba.
3. Győződjön meg róla, hogy mentette és a megnyitott queries.vql fájlon hagyta a fókuszt.
4. Nyomja meg a másik zöld nyíl gombot az ablakban a lekérdezések betöltéséhez.

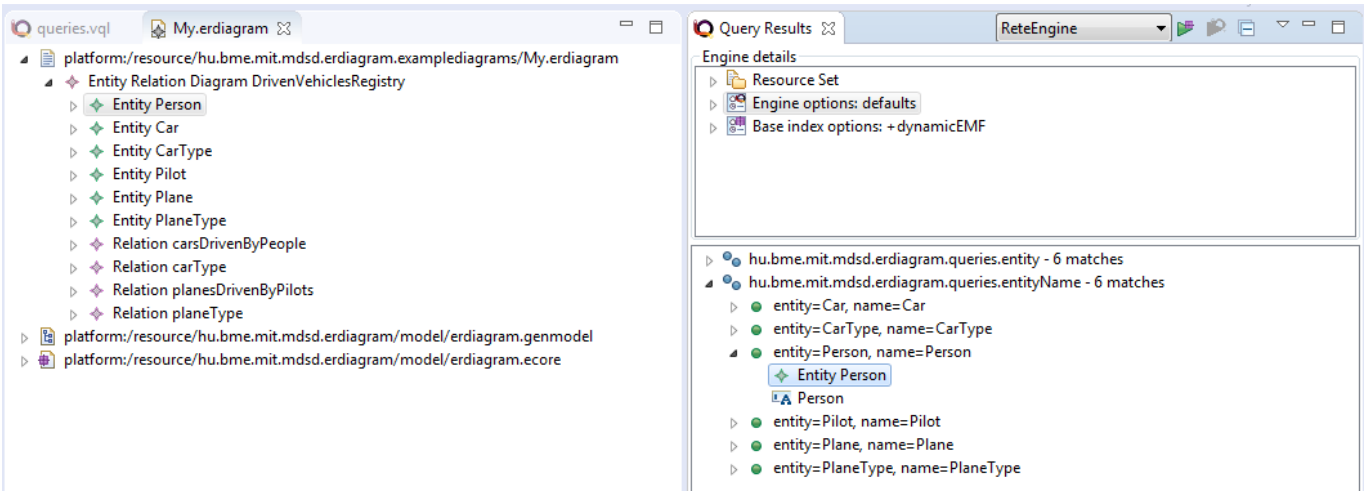


Az ablak felső paneljében megtekintheti a modellt és a Query engine konfigurációját (ehhez a gyakorlatoz a ReteEngine-t használja). A nézet alsó paneljében láthatja a lekérdezések találatait. Jelenleg nincsenek találatok

- ez azért van, mert a nézet nem ismeri a metamodelt (ecore modellt) és nem tudja összekapcsolni a modellt és a lekérdezéseket. Két lehetősége van itt:

1. Indítson egy futásidejű eclipse-et és töltse be az ott az példánymodellt és a lekérdezéseket (miután importálta a projektet a runtime workspace-be).
2. Vagy kapcsolja be a dinamikus módot, amely string-ekket használ, találatok kereséséhez a modellben: Window/Preferences/VIATRA/Query Explorer és pipálja be a dynamic EMF jelölőnégyzetet. Ezt a konfigurációt követően ki kell töltenie és újra betöltenie a modellt és a lekérdezéseket.

Most már láthatja az összes találatot. Tekintse át a kontextusmenüt is.



▶ Nézze meg a videót ▶

Mintanyelv (Pattern Language)

1. A nyelv megismerése érdekében írjunk néhány validációs lekérdezést, és jólformáltsági kényszert. Először hozzunk létre egy lekérdezést, amely ellenőrzi, hogy egy `NamedElement` neve csak egy üres karakterlánc-e:

```
pattern unnamedElement(element : NamedElement) {
    NamedElement.name(element, "");
}
```

Ez a minta azt mutatja, hogy a konstans attribútum értékeket literálok formájában közvetlenül a paraméterlista részében lehet megadni. Azonban amiket valóban szeretnénk elkapni, azok a NamedElement-ek, amelyeknek nincs nevük vagy üres a nevük:

```
pattern unnamedElement(element : NamedElement) {
    NamedElement.name(element, "");
} or {
    neg NamedElement.name(element, _);
}
```

Ehhez szükség volt egy diszjunkcióra (`or`) egy második minta törzzsel. Az új törzs elkapja a NamedElement típusú elemeket, amelyekhez egyáltalán nem tartozik olyan karakterlánc, amely kielégítené a `NamedElement.name` relációt - más szóval, ez az attribútum slot kitöltetlen. Egy mintaváltozó (mint `element`) vagy egy konkrét konstans literál (mint `""`) helyett itt az `_` karaktert használtuk, amely egy tetszőleges, nem meghatározott értéket jelöl; ezért a negált kényszer azt jelenti: "nem számít, melyik karakterláncra gondolsz, az nem az elem neve".

2. Hozzon létre egy lekérdezést, amely ellenőrzi, hogy a név nem nagybetűvel kezdődik:

```
pattern entityStartsWithLowerCase(entity) {
    Entity.name(entity, name);
    check (!name.matches("^[A-Z].+"));
}
```

Ez a minta mutatja a `check` blokkot, ahol széles körű *Xbase* kifejezéseket írhatunk (hasonlóan a Java-hoz). Ebben az esetben egy reguláris kifejezést definiálunk. Egy lehetséges alternatíva lett volna `check(name.toFirstUpper != name)` - ez demonstrálja az Xbase kiterjesztési metódusokat is.

3. Hozzon létre egy lekérdezést, amely ellenőrzi, hogy két entitásnak azonos-e a neve:

```
pattern entityNameAmbiguity(entity1, entity2, commonName) {
    Entity.name(entity1, commonName);
    Entity.name(entity2, commonName);
    entity1!=entity2;
}
```

Ez a minta a `!=` (*nem egyenlő*) operátort mutatja be, amely kiválaszt a két különböző entitást az előfordulási modellből. Lehet használni a `==` operátort az egyenlőség teszteléséhez is, de ebben az esetben egyszerűen ugyanazt a változót, a `commonName`-t kell kétszer használni.

Vegyük figyelembe, hogy ezen a módon megfogalmazva a lekérdezés nem hatékony, mivel összehasonlítja az összes lehetséges entitást, amelynek költsége négyzetes a modell méretében. A hatékonyabb (lineáris költségű) megoldás az lenne, ha megszámolnánk az egyes entitások neveit, majd azokat az entitásokat jelentenénk, amelyek neveit többször számoltuk:

```
pattern entityNameAmbiguity(entity : Entity, name : java String) {
    Entity.name(entity, name);
    howManyWithSameName == count Entity.name(_, name);
    check(howManyWithSameName > 1);
}
```

A `count` módosítót használtuk, amely példa egy aggregátorra. Ez egyetlen értéket aggregál a hozzá kapcsolódó mintakényszerek különböző lehetséges illeszkedéseiből. Vannak más aggregátorok is, például `sum`, `max`, stb. Hasonlóan van a korábban használt speciális módosító, a `neg`, amely a kapcsolódó kényszernek az összes illeszkedésére ellenőrzi, hogy nincs-e egyezés.

4. Az attribútumoknak általában nem kell, hogy egyedi neveik legyenek, de egyetlen entitás attribútumainak igen. Tehát az azonos nevű attribútumok kétértelműségének tesztelése attribútumnevek megszámlálását igényli ugyanazon entitáson belül, amihez egy segédmintára van szükségünk:

```
pattern attributeNameAmbiguity(attribute : Attribute, name : java String) {
    find attributeQualifiedName(entity, attribute, name);
    howManyWithSameName == count find attributeQualifiedName(entity, _,
name);
    check(howManyWithSameName > 1);
}
pattern attributeQualifiedName(entity : Entity, attribute : Attribute, name
:    java String) {
    Entity.attributes(entity, attribute);
    Attribute.name(attribute, name);
}
```

5. Az előző lekérdezések jólformáltsági kényszerek voltak, amelyek, talán meglepő módon, azon eseteket találták meg, amikor a modell rosszul van formázva. Most hozzunk létre egy lekérdezést, amely kifejezetten a jólformált elemeket keresi.

   Ez a minta bemutatja, hogyan lehet korábban meghatározott mintákat újra felhasználni almintaként. Ehhez használjuk a `find` kulcsszót, majd megadjuk az alminták azonosítóját és végül hozzáadjuk a változókat. (A `_`-vel kezdődő változók nem "fontosak", ezért nem használhatók más sorokban a mintában).

   Ismét használjuk az `or` kulcsszót, amely azt jelenti, hogy a minta illeszkedik, ha az első *vagy* a második *vagy* a harmadik *vagy* stb. törzse illeszkedik.

```
pattern badEntity(entity) {
    find emptyNamedElement(entity);
} or {
    find entityStartsWithSmallCase(entity);
} or {
    find sameNamedEntities(entity, _, _);
}
```

   A teljes modell jólformált, (az entitásokra vonatkozóan legalábbis) ha nincsenek olyan entitások, amelyek megfelelnek bármely rosszul formáltsági hibamintának, azaz nincs illeszkedése a `badEntity` lekérdezésnek. Ehhez használhatjuk a már jól ismert `neg` kulcsszót, amelyet ezúttal a `find` kulcsszóval kombinálunk.

```
pattern wellFormedEntites() {
    neg find badEntity(_);
 }
```

6. Következő lépésként hozzunk létre egy jólformáltsági kényszert a `Relation`-ökre, amely ellenőrzi, hogy mindkét `RelationEnding`-je megvan-e. Ehhez szükségünk lesz egy segéd mintára.

```
// Legalább egy pozitív kényszer szükséges a változón - ebben az esetben r
egy Relation:
pattern relationWithoutEnding(r : Relation) {
    neg Relation.leftEnding(r, _);
} or {
    neg Relation.rightEnding(r, _);
}
pattern wellFormedRelation() {
    N == count find relationWithoutEnding(_);
    N == 0;
}
```

Jegyezzük meg, hogy a `neg` használata megegyezik a `count` kulcsszó használatával és biztosítja, hogy az értéke nulla legyen.

7. Az entitás attribútumainak számát is lekérdezhetjük:

```
pattern attributeCount(e: Entity, num: java Integer) {
    Entitiy(e); // opcionális, hiszen már a paraméter sorban meg van adva
    num == count Entity.attributes(e, _);
}
```

8. Keressük meg az entitást, amelyik az ábécében először szerepel:

```
pattern hasBiggerName(e1, e2) {
    Entity.name(e1, name1);
    Entity.name(e2, name2);
    check(name1 > name2);
}

pattern firstEntity(e : Entity) {
    neg find hasBiggerName(e, _);
}
```

Az előbbi technika hasznos lehet akkor is, ha két illeszkedés pontosan ugyanaz, csak a paraméterek sorrendje különbözik. Az `check(name1 > name2);` kifejezés meghatározza a sorrendet, így csak az egyik illeszkedés érhető el az eredményhalmazban. Újra hangsúlyozzuk, hogy akárcsak a név egyezés példában, ez a megoldás négyzetes, és lineárisra kellene átszervezni (az aggregáló `min` használatával).

9. Keressük meg az entitás összes ősentitását a tranzitív lezártatt használva.

```
pattern directSuperEntity(e, superEntity) {
    Entity.isA(e,superEntity);
}

pattern transitiveSuperEntity(e, superEntity) {
    find directSuperEntity+(e, superEntity);
}
```

Validáció

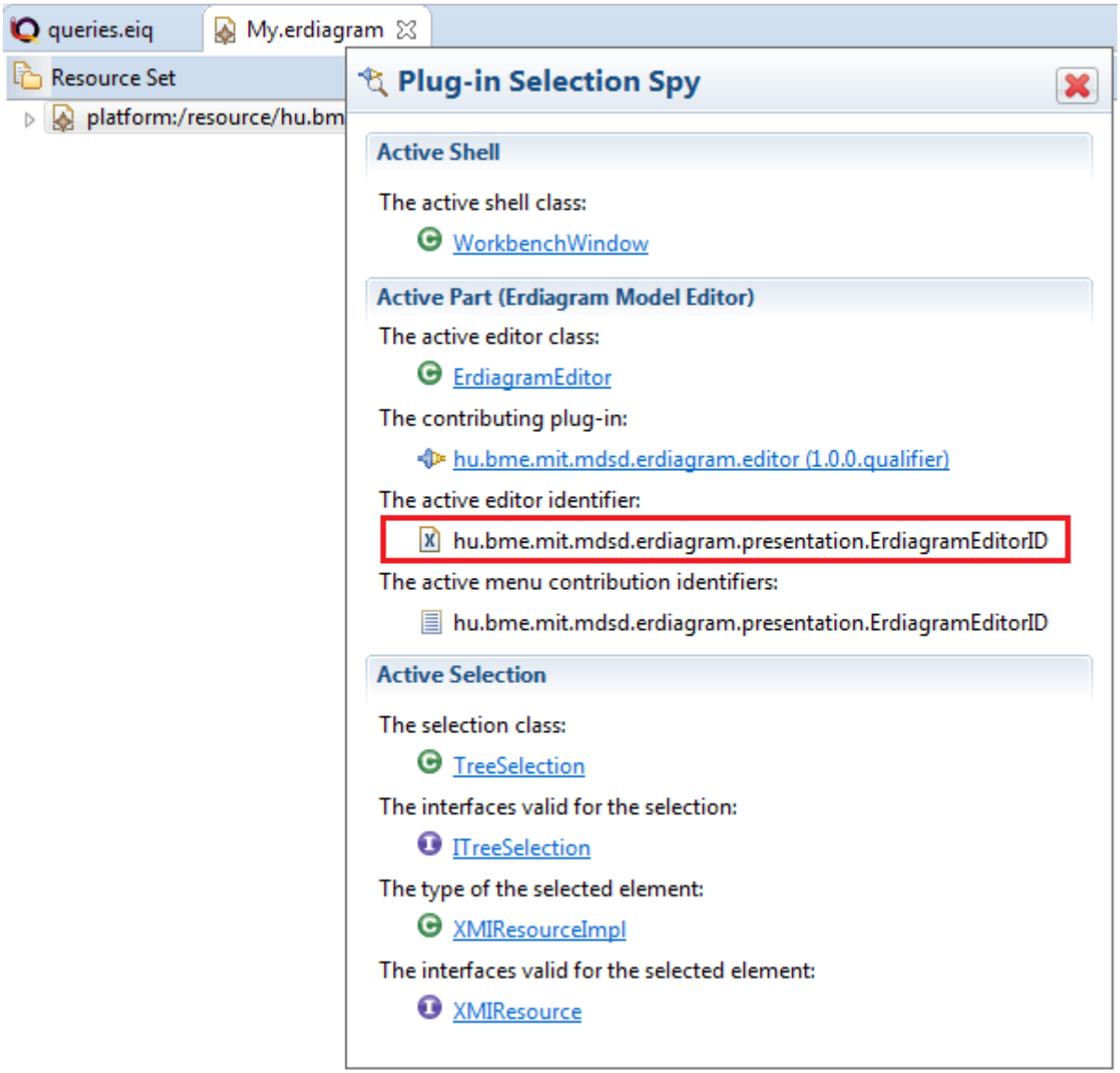
A VIATRA lehetőséget biztosít az EMF modell-eken végzett validációhoz a keretrendszer mintái alapján. Ezeket a szabályokat különböző EMF példánymodelleken ki lehet értékelni, és a kényszerek megsértése esetén a hibák jelzése automatikusan bekerül az Eclipse Problems nézetébe.

A **@Constraint** annotáció használható arra, hogy egy mintát validációs szabállyként használjunk. Ha a keretrendszer legalább egy mintát talál az ilyen annotációval, akkor egy `.validation` projektet hoz létre.

Az annotáció paraméterei:

- *key:* Az a paraméter, amelyhez a megsértés jelzése csatolva lesz.
- *message:* Az üzenet, amelyet a megsértés esetén megjelenít. Az üzenet hivatkozhat a paraméterváltozókra $ jelek között, vagy az EMF tulajdonságaira, például a $Param1.name$ formában.
- *severity:* "warning" vagy "error"
- *targetEditorId:* Az Eclipse szerkesztőazonosító (editor ID), amelyhez a validációs keretrendszer regisztrálja magát a kontextusmenüben.

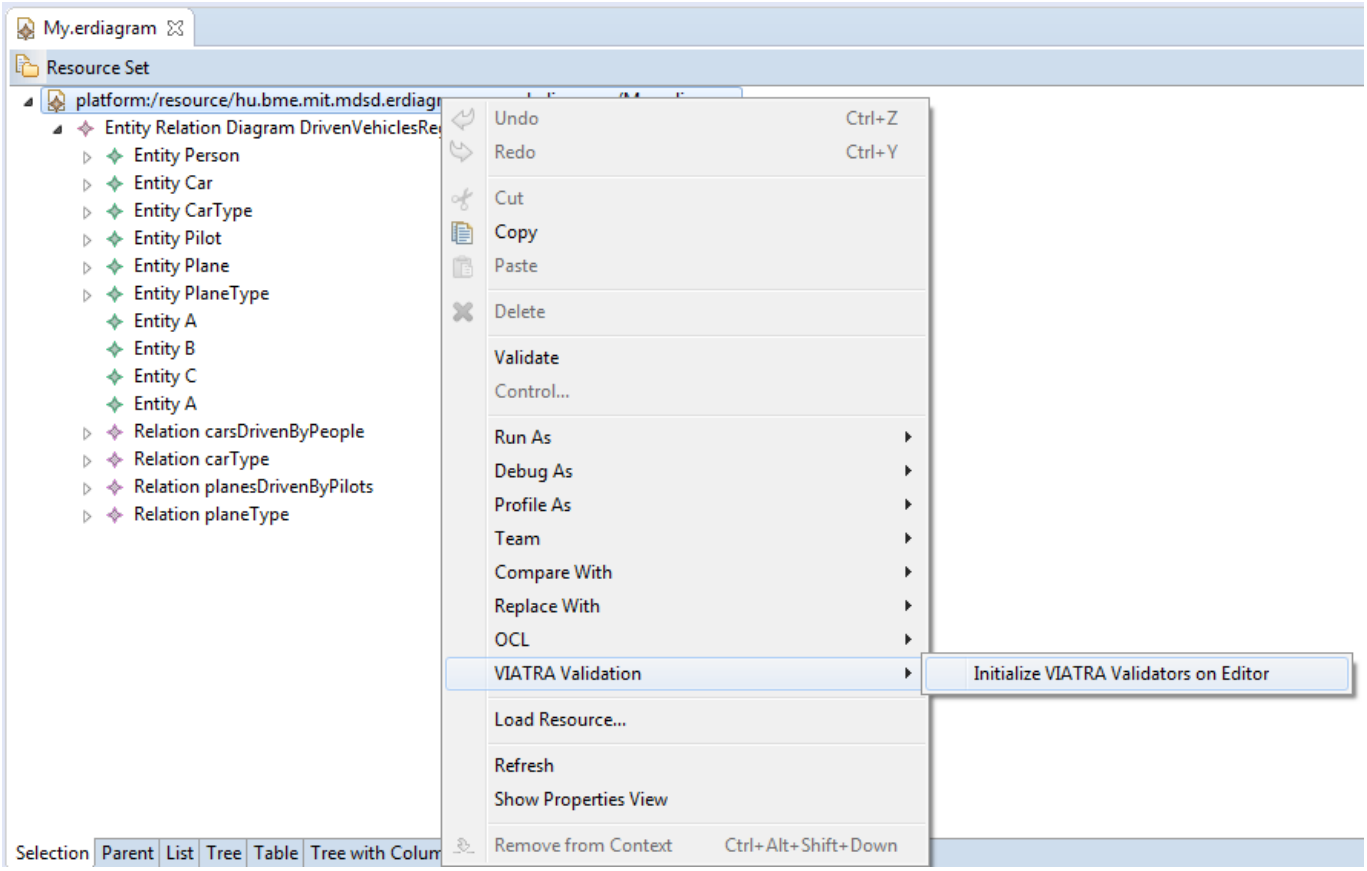
A megfelelő szerkesztőazonosító megtalálásához a *Plug-in Selection Spy* eszköz használható a *SHIFT* + *ALT* + *F1* billentyűkombinációval. Vagy ellenőrizhetjük a plugin.xml fájlt a `hu.bme.mit.mdsd.erdiagram.editor` projektben.



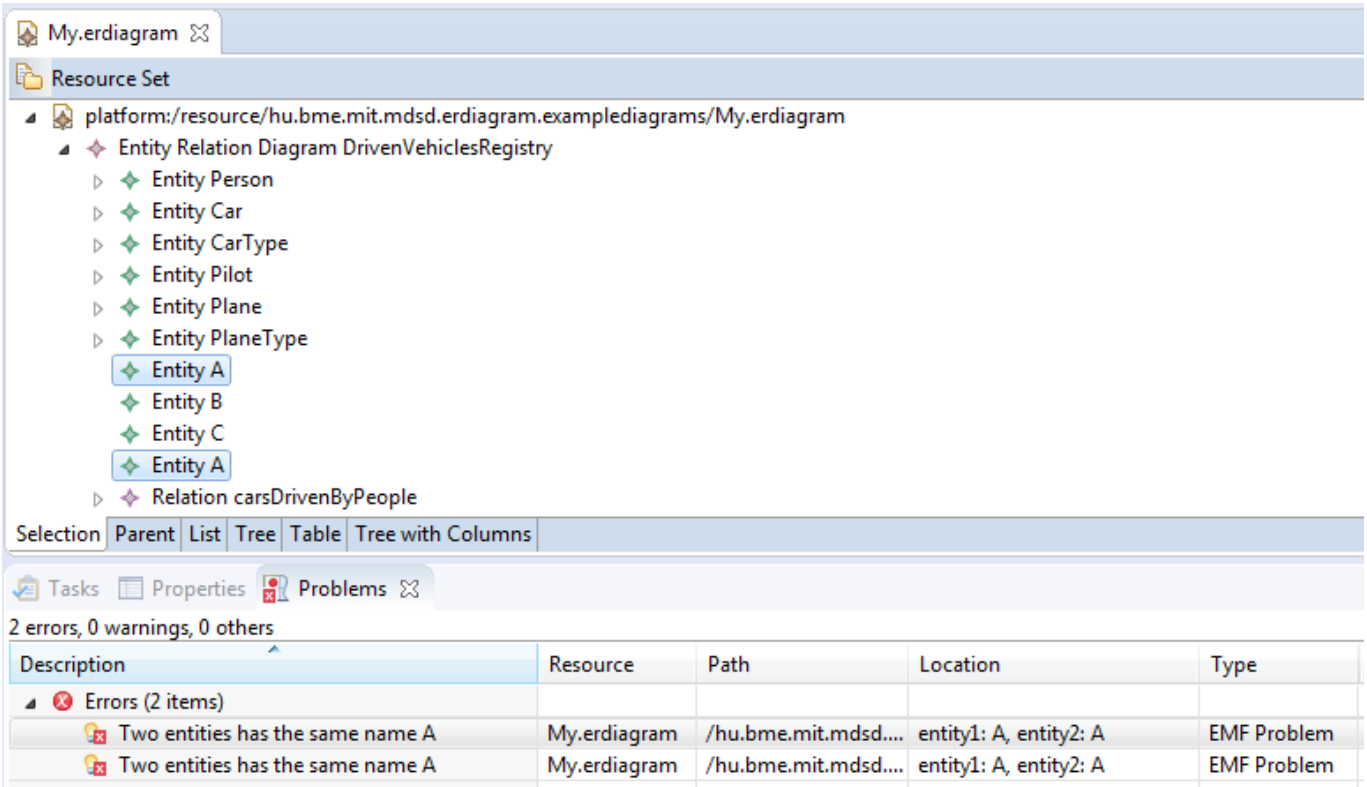
Hozzunk létre egy validációs szabályt, a `sameNamedEntities` minta felhasználásával (szándékosan az első verziót használva):

```
@Constraint(
    key = {entity1, entity2},
    severity = "error",
    message = "Két entitás azonos nevet használ: $commonName$.",
    targetEditorId = "hu.bme.mit.mdsd.erdiagram.presentation.ErdiagramEditorID"
)
pattern sameNamedEntities(entity1, entity2, commonName) {
    Entity.name(entity1, commonName);
    Entity.name(entity2, commonName);
    entity1 != entity2;
}
```

Build után egy `.validation` plugin projekt generálódik. Indítsunk egy futó Eclipse-t, hogy telepíthessük a validációs plugint. A modell megnyitása után válasszuk ki a **VIATRA Validation | Initialize VIATRA Validators on Editor** lehetőséget.



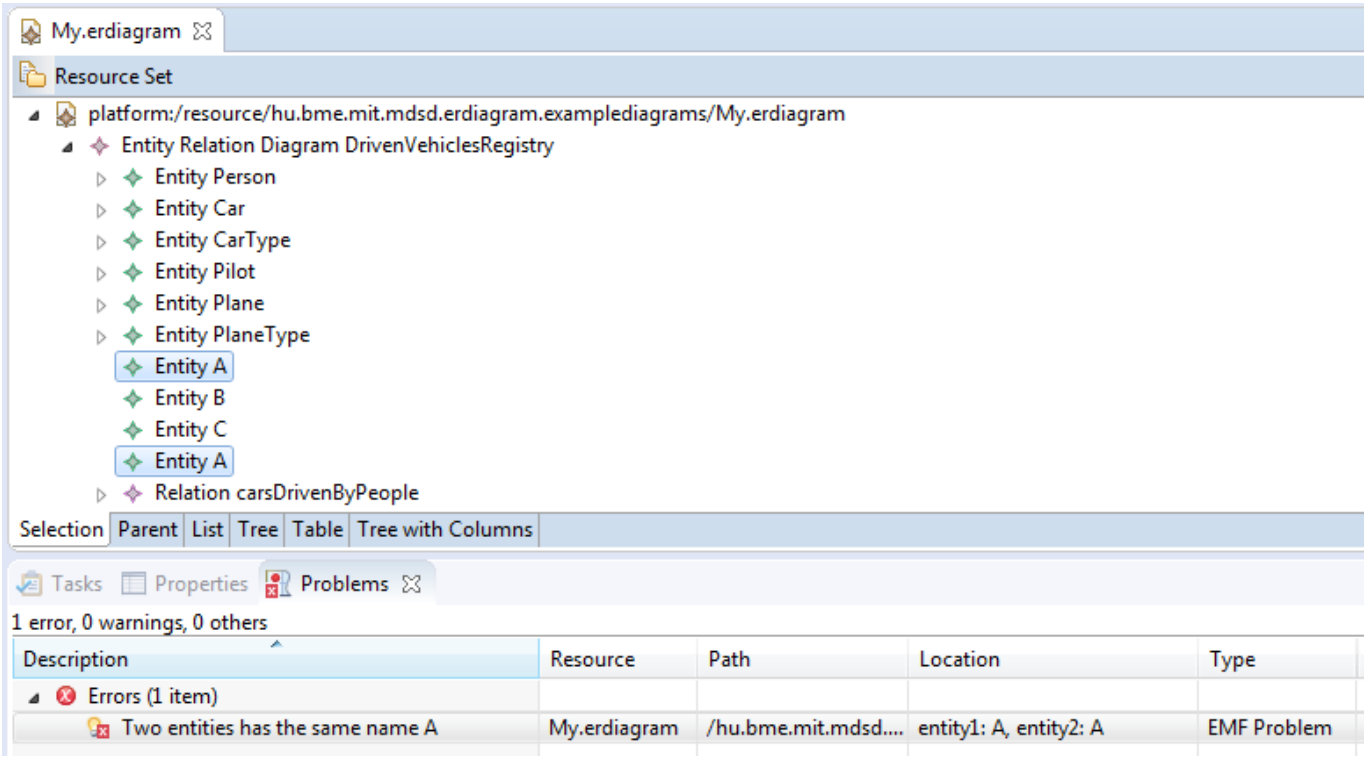
A következő hibák kellenek, hogy megjelenjenek (ha duplán kattint a hibára, akkor kijelöli a problémás EClasses elemeket):



A két hiba ugyanazt az hibát jelöli. Hogy ezt megoldjuk, adjuk hozzá a következő paramétert a kényszer annotációhoz:

```
symmetric = {entity1, entity2}
```

Ezután működni fog, ahogy kell:



Természetesen ez a szimmetriaprobléma nem merül fel a `count`-alapú megoldással.

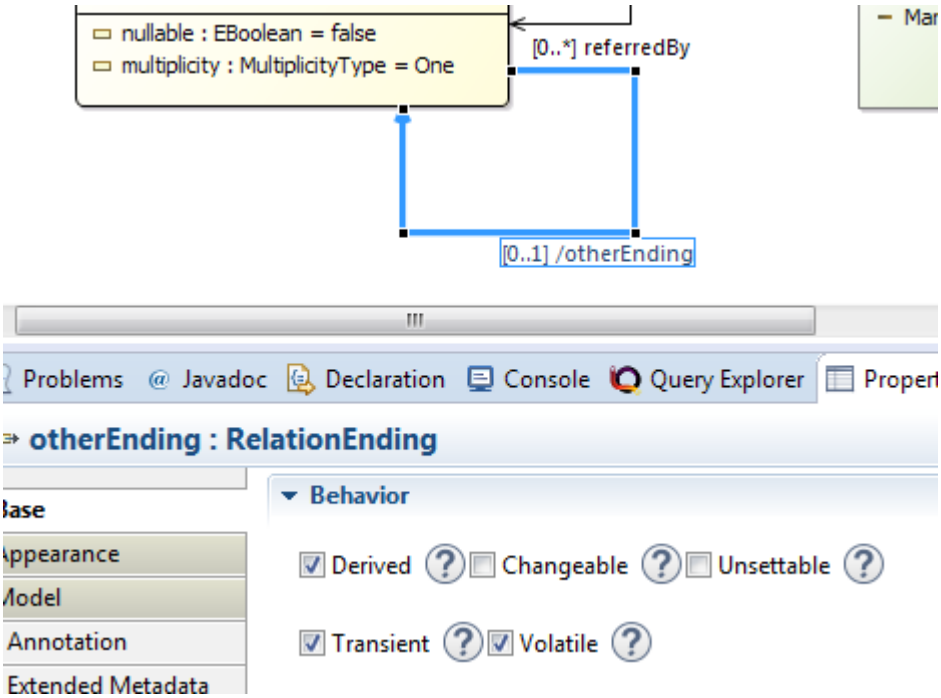
További érdekes példákat találhat itt: https://github.com/ftsrg/mdsd-examples/blob/vql_2020/hu.bme.mit.mdsd.erdiagram.queries/src/hu/bme/mit/mdsd/erdiagram/queries/validation.vql

Derived feature (Számított értékek)

Hozzunk létre egy derived feature-t a `RelationEnding`-ek között, amely visszaadja a szülő `Relation` másik `RelationEnding`-jét. Ehhez a következő három lépésre van szükség:

1. Definiáljon egy egyirányú relációt `RelationEnding`-ek között [0..1] multiplicitással, és nevezze el `otherEnding`-nek. A feature következő attribútumait állítsuk be:

- derived = true (jelezve, hogy a jellemző értéke a modellből származik)
- changeable = false (távolítsuk el a setter metódusokat)
- transient = true (a fájlban történő mentés elkerülése érdekében)
- volatile = true (hogy eltávolítsuk a meződeklarációt az objektumból)



Ne felejtsük el menteni a modellt, újratölteni a genmodellt és újragenerálni a modellkódot!

1. A VIATRA támogatja az hatékony, inkrementálisan karbantartható és jól viselkedő származtatott értékek meghatározását az EMF-ben, haladó modell lekérdezések és az inkrementális értékelés használatával, amely kiszámítja a származtatott feature-ök értékét, valamint automatikusan létrehozza a kódot a meglévő alkalmazásokba való integráláshoz.

A **@QueryBasedFeature** annotáció használható arra, hogy egy mintát derived feature-ként jelöljünk meg. Ha a keretrendszer képes megtalálni a feature-t a minta szignatúrájából (*minta név*, *első paraméter típusa*, *második paraméter típusa*), az annotáció paramétereit üresen hagyhatjuk.

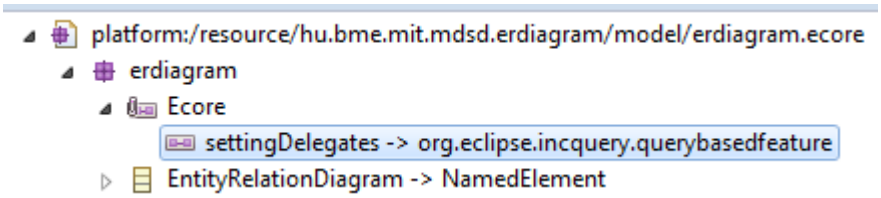
Az annotáció paraméterei:

- feature ="featureNév" (alapértelmezett: minta név) - jelzi, hogy mely származtatott értéket határozza meg a minta
- source ="Src" (alapértelmezett: első paraméter) - jelzi, hogy mely query paraméter (a nevét használva) az forrás EObject, az ezen paraméter kikövetkeztetett típusa jelzi, hogy mely EClass generált kódját kell módosítani
- target ="Trg" (alapértelmezett: második paraméter) - jelzi, hogy mely query paraméter (a nevét használva) a származtatott jellemző célpontja
- kind ="single/many/counter/sum/iteration" (alapértelmezett: feature.isMany?many:single) - jelzi, hogy milyen számítást kell elvégezni a query eredményeken, hogy azokat származtatott jellemző értékekhez lehessen hozzárendelni
- keepCache ="true/false" (alapértelmezett: true) - jelzi, hogy külön gyorsítótárat kell-e tárolni az aktuális értékkel. A Single és Many jellegű származtatott feature-ök dolgozhatnak külön gyorsítótár nélkül, mivel a VIATRA RETE hálózat már tartalmazza az aktuális értékek gyorsítótárát

  Hozzunk létre egy mintát, amely megtalálja a másik végpontot, majd jelöljük meg a `QueryBasedFeature`-vel:

```
@QueryBasedFeature
pattern otherEnding(ending : RelationEnding, other : RelationEnding) {
    Relation.leftEnding(relation, ending);
    Relation.rightEnding(relation, other);
} or {
    Relation.rightEnding(relation, ending);
    Relation.leftEnding(relation, other);
}
```

Mentse el és build-elje újra. A VIATRA módosítani fogja az Ecore modellt egy annotációval.



1. Töltse be újra az Ecore modellt a genmodel számára, majd újra generálja a modellkódot. Most, ha a `relationEnding.getOtherEnding()`-et használja a modellen, helyesen visszaadja a `RelationEnding` objektumot. Vegye figyelembe, hogy szüksége lesz további VIATRA inicializációs kódra, vagy futtathatja JUnit Plug-In tesztben. Úgy is ellenőrizheti azt, hogy működik-e, ha egy runtime eclipse-ben használja a generált fa szerkesztőt.

Számított attribútum címkézésként

Tegyük fel, hogy az példánymodell szerkesztő fában szeretnénk megjeleníteni a relációk végpontjait olyan címkékkel, amelyek mind a nevüket, mind pedig típusukat (a `target` Entitás nevét) jelzik. Jelenleg nincs egyetlen olyan címke feature, amely betölti ezt a szerepet, ezért egy származtatott attribútumot kell definiálnunk erre a célra. Követhetjük az előzőhöz hasonló eljárást:

1. Definiáljunk egy `EString` típusú, `endingLabel` nevű EAttribute-ot a `RelationEnding` EClass-ban, [0..1] (alapértelmezett) multiplicitás értékkel. Állítsuk be az alábbi tulajdonságokat, mint az előbb:

- derived = true (azt jelzi, hogy a tulajdonság értéke a modellből számítódik)
- changeable = false (eltávolítja a setter metódusokat)
- transient = true (elkerüljük az érték fájlba mentését)
- volatile = true (eltávolítja a meződeklarációt az objektumból)

1. Ne felejtsük el menteni a modellt, újratölteni a genmodelt és újra generálni a modellek kódját!

2. Opcionális kísérlet: próbáljuk meg manuálisan megvalósítani ezt a származtatott tulajdonságot, a `RelationEndingImpl.getEndingLabel()` placeholder metódus kitöltésével, olyan kóddal, amely `getName()` és `getTarget().getName()` eredményekből összeállít egy String eredményt. (Hagyjuk `@generated`-nek, így automatikusan kicserélődik a valódi megoldással.) Állítsuk be ezt a tulajdonságot címkézésként (label feature) a .genmodelben, és generáljuk újra a model/edit/editor kódot. Indítsunk egy runtime Eclipse-et és figyeljük meg, hogy a látható `RelationEnding` címkék nem fogják tükrözni a cél Entitás típusának átnevezését. Ez azt mutatja, hogy egy ilyen megközelítés egy zsákutca.

3. Adjuk hozzá az alábbi VQL kódot a query projekthez:

```
@QueryBasedFeature(keepCache = false)
pattern endingLabel(ending: RelationEnding, label: java String) {
    find nameLabel(ending, nameLabel);
    find typeLabel(ending, typeLabel);
    label == eval (String.format("%s : %s", nameLabel, typeLabel));
}

pattern nameLabel(elem: NamedElement, label: java String) {
    NamedElement.name(elem, label);
    label != "";
} or {
    NamedElement.name(elem, "");
    label == "<unnamed>";
} or {
    neg NamedElement.name(elem, _);
    label == "<unnamed>";
}

pattern typeLabel(elem: RelationEnding, label: java String) {
    RelationEnding.target.name(elem, label);
    label != "";
} or {
    RelationEnding.target.name(elem, "");
    label == "<untyped>";
} or {
    neg RelationEnding.target.name(elem, _);
    label == "<untyped>";
}
```

Mentsük és build-eljük. A VIATRA módosítja az Ecore modellt egy annotációval.

1. Töltsük be újra az ecore modellt a genmodelhez. Válasszuk ki a `RelationEnding` elemet a genmodelben, és állítsuk be a "label feature" tulajdonságát (az "Edit" kategóriában) az új származtatott attribútumra, az `endingLabel`-ra. Mentés után generáljuk újra a model és edit és editor kódot.

2. Élvezzük az új fa szerkesztőt a szebb címkékkel a reláció végekhez 😃

Néhány További Lekérdezés

Az alábbi lekérdezések az `isA` kapcsolatok érdekes következményeit számolják ki. Az alábbi kód részletek az előzőleg definiált segítőmintákat használják, mint például `find transitiveSuperEntity(entity, superEntity);`, azonban közvetlenül a `Entity.isA*(entity, superEntity);` írása is ugyanolyan jól működik.

1. Készítsen egy mintát, amely a típus hierarchiában észlel egy kört:

```
pattern circleInTypeHierarchy(entity) {
    find transitiveSuperEntity(entity, entity);
}
```

2. Készítsen egy mintát, amely egy (tranzitív) rombuszt észlel a típus hierarchiában. Biztosítsa, hogy ne legyen több mint egy azonos rombuszt reprezentáló illeszkedés:

```
pattern diamondInTypeHierarchy(entity1, entity2, entity3, entity4) {
    find transitiveSuperEntity(entity1,entity2);
    find transitiveSuperEntity(entity1,entity3);
    find hasBiggerName(entity2, entity3); //entity2 != entity3;
    find transitiveSuperEntity(entity2,entity4);
    find transitiveSuperEntity(entity3,entity4);
}
```

3. Bővítse a mintákat, hogy az örökölt kapcsolatokat és attribútumokat is beolvassák:

```
pattern attribute(entity, attribute) {
    Entity.attributes(entity,attribute);
} or {
    find transitiveSuperEntity(entity, superEntity);
    find attribute(superEntity, attribute);
}
```

és

```
pattern relation(entity1, entity2) {
    Relation.leftEnding.target(relation, entity1);
    Relation.rightEnding.target(relation, entity2);
} or {
    find transitiveSuperEntity(entity1, superEntity);
    find relation(superEntity, entity2); // rekurzió - nem szükséges itt,
csak a demo miatt
}
```

Ezeket a projekteket ebben a repository-ban találja a `VQL` branch-en.

Referenciák

- Útmutató: https://www.eclipse.org/viatra/documentation/tutorial.html
- Pattern nyelv: https://www.eclipse.org/viatra/documentation/query-language.html
- Validation Framework, Query-based Features: https://www.eclipse.org/viatra/documentation/addons.html
- Példák repója: https://github.com/ftsrg/mdsd-examples