FeladatkiÍrás

Ahogy a modellalapú szoftverfejlesztést az ipari gyakorlatban egyre nagyobb és komplexebb rendszerek tervezésében használják, a jelenlegi modellezőeszközök gyakran ütköznek skálázhatósági korlátokba. Emiatt kutatási és fejlesztési szempontból kiemelt jelentőségű, hogy a szoftvermodellezés számára olyan technológiákat, algoritmusokat és eszközöket fejlesszünk ki, amelyek biztosítják az eszközök megfelelő hatékonyságát és használhatóságát nagyon összetett rendszerek modellezése esetén is - ez a kulcsa annak, hogy a módszertan elismert előnyeit a fejlesztői produktivitás, szoftverminőség és karbantarthatóság tekintetében a jövőben is folyamatosan ki lehessen aknázni. A MONDO EU FP7 kutatási projekt célja a fent említett skálázhatósági kihívások megoldása mind elméleti, mind gyakorlati oldalról, az elosztott rendszerek és algoritmusok módszereire, illetve napjaink felhő alapú infrastruktúrájára alapozva. A skálázhatóság elérése érdekében a szoftvermodellezési környezetben elsősorban nagy gráfmodellek és komplex szakterületi modellező nyelvek szisztematikus kezelését kell megoldani.

A hallgató feladata, hogy a tanszéken folyó kutatásba bekapcsolódva megtervezzen és megvalósítson az offline kollaboratív modellező eszköz szerver oldali prototípusát, illetve egy kliens oldali változás követő rendszert, amely képes nyomonkövetni a modelleken végzett programmatikus illetve kézi módosításokat is, nyomonkövethetőségi modellek formájában sorosítva azokat.

A szakdolgozat kidolgozása a következő részfeladatok megoldását igényli:

* Végezzen irodalomkutatást, és mutassa be a jelenleg elérhető kollaboratív keretrendszereket.
* Tervezze meg az offline kollaboratív modellező keretrendszer szerveroldali alkalmazásának interfészét, és valósítsa is meg a prototípust.
* Tervezze és valósítsa meg a modelleken végzett változásokat automatikusan követő rendszert.
* Egy részletesen kidolgozott példaalkalmazáson keresztül mutassa be a komplett rendszer működését, használatát és a fejlesztés menetét. Értékelje munkáját és vázolja fel a továbbfejlesztési lehetőségeket.

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

**Budapesti Műszaki és Gazdaságtudományi Egyetem**

Villamosmérnöki és Informatikai Kar

Kollaboratív modellező keretrendszer fejlesztése

Készítette

Konzulens

2014

Tartalomjegyzék

[Összefoglaló 6](#_Toc405338400)

[Abstract 7](#_Toc405338401)

[1. Bevezetés 8](#_Toc405338402)

[2. Háttér technológiák 9](#_Toc405338403)

[2.1. Eclipse Modelling Framework 9](#_Toc405338404)

[2.1.1. Ecore 9](#_Toc405338405)

[2.1.2. Resource 10](#_Toc405338406)

[2.1.3. Notifikáció 10](#_Toc405338407)

[3. Fejezet 12](#_Toc405338408)

[3.1. Kollaboráció modellező rendszerekben 12](#_Toc405338409)

[3.2. MONDO kollaborációs keretrendszer 12](#_Toc405338410)

[3.3. Az offline rendszer műveletei 14](#_Toc405338411)

[3.3.1. Checkout 14](#_Toc405338412)

[3.3.2. Merge 14](#_Toc405338413)

[3.3.3. Update 14](#_Toc405338414)

[3.3.4. Commit 14](#_Toc405338415)

[3.3.5. Zár hozzáadása 14](#_Toc405338416)

[3.3.6. Zár levétele 15](#_Toc405338417)

[3.3.7. Zárak listázása 15](#_Toc405338418)

[3.3.8. Napló 15](#_Toc405338419)

[3.3.9. Jogosultság hozzáadása 15](#_Toc405338420)

[3.3.10. Jogosultság elvétel 15](#_Toc405338421)

[3.3.11. Jogosultságok listázása 15](#_Toc405338422)

[4. Fejezet 17](#_Toc405338423)

[4.1. Nyomonkövetés megvalósítása 17](#_Toc405338424)

[4.2. Nyomonkövetési metamodell 18](#_Toc405338425)

[4.2.1. A rendszer fő elemei 18](#_Toc405338426)

[4.2.2. A megvalósított műveletek 19](#_Toc405338427)

[5. Fejezet 23](#_Toc405338428)

[5.1. A workspace tracker architektúrája 23](#_Toc405338429)

[5.2. A trace modell elkészítése 24](#_Toc405338430)

[5.3. MyContentAdapter 24](#_Toc405338431)

[5.3.1. Event types 25](#_Toc405338432)

[5.3.2. Példánymodell elmentése 25](#_Toc405338433)

[6. Motivációs példa 26](#_Toc405338434)

[6.1. Szélturbina 26](#_Toc405338435)

[Összefoglalás 27](#_Toc405338437)

[Köszönetnyilvánítás 28](#_Toc405338438)

[Ábrák jegyzéke 29](#_Toc405338439)

[Táblázatok jegyzéke 30](#_Toc405338440)

[Irodalomjegyzék 31](#_Toc405338441)

[Függelék 32](#_Toc405338442)

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, 2014. 12. 02.

Vikár András

# 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

Napjainkban egyre nagyobb hangsúlyt kap a modellalapú szoftverfejlesztés. Egyre nagyobb és komplexebb rendszerek tervezésében használják, viszont a jelenlegi modellező rendszerek gyakran ütköznek skálázhatósági problémákba. A félév során becsatlakoztam a MONDO EU FP7 kutatási projektbe, mely pontosan az ilyen rendszerek skálázhatósági problémáinak megoldására törekszik.

A csoportnak, mellyel együtt dolgoztam egy kollaborációs keretrendszer megalkotása volt a célja. A rendszer mind az online, mind az offline felhasználási területen biztosítja a modellek több felhasználó általi szerkesztését, módosítását. Én az offline kollaborációs rendszer fejlesztésébe kapcsolódtam bele. Feladatom, hogy elkészítsek egy modellt, mely követni tudja a modelleken véghezvitt módosításokat.

# Háttér technológiák

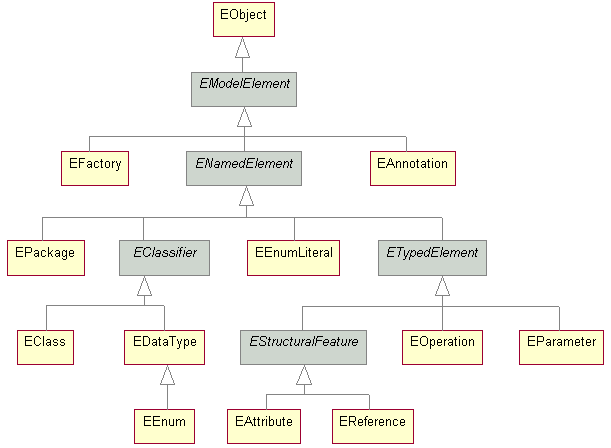
## Eclipse Modelling Framework

Napjaink egyik legelterjedtebb, modell alapú architektúra fejlesztő eszköze az Eclipse Modelling Framework (EMF). A keretrendszerrel metamodelleket lehet készíteni, melyek segítségével a rendszer architektúra könnyebben áttekinthetővé válik. Valamint kódgenerálással segíti a kódolási folyamatot. Sorosításhoz XMI-t használ.

### Ecore

EMF támogatott nyelvek közül a legfontosabb, legtöbbet használt. Metamodell definiálása, példánymodell létrehozása, validálás.

Ecore felépítése:



fontosabb elemek:

EObject: Minden Ecore osztály őse

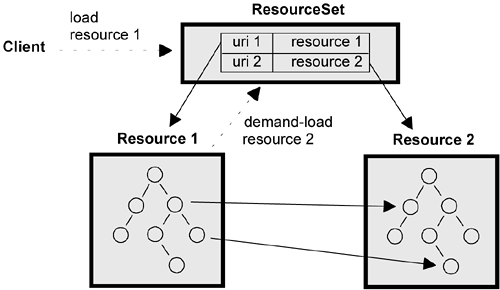
EClass: Osztályt reprezentál, neve, attribútumai, referenciái vannak.

EAttribute: Attribútumot reprezentál. Neve típus és értéke van. Számosságát szabályozni lehet.

EReference: 2 osztály közötti egyirányú asszociációt reprezentál, neve, számossága van, megadható, hogy tartalmazást valósít-e meg.

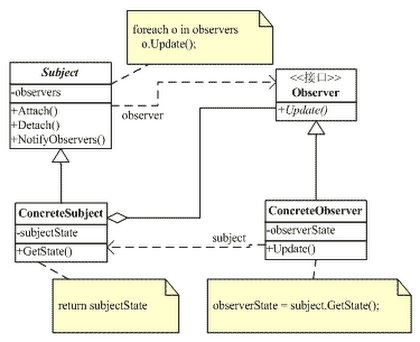
### Resource

Az EMF egy példánymodelt egy Resource-ben tárol, A resourceben található a fa struktúrájú példánymodell. A resourceokat a keretrendszer resourceSetben tárolja, melyben az összetartozó modellek kerülhetnek.



### Notifikáció

Mind az Eobject minda Resource, minda ResourceSet megvalósítja a Notifier interfacet, melynek, eAdapter függvénye visszaadja az objektumhoz tartozó adaptereket. A generált kódban a példánymodell fel van készítve a különböző elemi változásokra, ezek bekövetkeztekor, meghívja az adott objektum minden adapterének a megfelelő függvényét, mely a változásra reagál. A változás adatait egy Notification interfészt megvalósító osztály tartalmazza, melyet az adapter paraméterül megkap.



# Fejezet

Kollaborációs rendszer

## Kollaboráció modellező rendszerekben

Napjainkban a modellek bonyolultsága és mérete egyre tovább nő. Szükség van tehát több felhasználói keretrendszerre (mely a hagyományos forráskód alapú fejlesztői környezetekben már megszokott) kollaborációval, és a hozzá kapcsolódó technológiákkal, mint a verziókezelés, konfliktuskezelés, a modell migráció és egyesítés (merging). Sajnos a tudomány jelenlegi állásában a modellező technológiák még nem zárkóztak fel jelenlegi eszközökhöz, sem funkciókban, sem érettségben. Ezért a fejlesztőeszköz készítőknek alkalmi megoldásokat kell megvalósítaniuk.

Léteznek modellezett eszközökhöz készített tároló rendszerek, melyek leginkább a perzisztenciára és a több infrastruktúrán való egyidejű hozzáférésre (cliens-server) összpontosítanak. Ezek az eszközök csak korlátozottan támogatják az olyan eszközöket, mint a konfliktuskezelés, a brench-ek használata, valamint a modell összehasonlítás. Ehelyett kiterjesztett mechanizmusokat és központi API-kat nyújtanak, hogy kiegészítő, funkció specifikus eszközöket lehessen használni. A következőkben 3 féle technológiát mutatok be:

* Eclipse Modeling Framework Connected Data Objects: Az EMF CDO keretrendszer egy model repository EMF modellek tárolásához, elsősorban a perzisztenciát és a verziókezelést célozva meg. A rendszerben egy egyszerű kollaboratív hozzáférési réteg található. A CDO megvalósít egy egyedi, tranzakció menedzsmentre alapuló API-t, valamint nézeteket az implivit zárak kezeléséhez. Sajnos nincs kiforrott támogatás a konfliktuskezeléshez és az egyesítéshez (merging)
* Dawn project: A CDO kollaborációs eszközének javítására készített Dawn projekt elsődleges célja, hogy elkülönüljön a kollaborációs UI és a hozzá tartozó API. A cél, hogy létrehozzanak előzetes kollaborációs primitíveket (zárkezelés, update, commit) az EMF és GMF editorok számára.
* MORSA: A MORSA egy újabb technológia skálázható modell perzisztenciára EMF modelleken. A rendszer NoSQL háttér technológián és on-demand töltési és gyorsítótárazási technológiákra alapul. Az elsődleges hangsúly a skálázhatóságon van, és csak előzetes lekérdezéseket és integrációs lehetőségeket biztosít.

## MONDO kollaborációs keretrendszer

A szakdolgozatom készítéséhez bekapcsolódtam a MONDO EU FP7 kutatási projektbe. A projekt célja, hogy a modellalapú szoftverfejlesztés skálázhatósági problémáira megoldást találjon, az elosztott rendszerek és algoritmusok módszereire, illetve napjaink felhő alapú infrastruktúrájára alapozva. A projekten belül az egyik kutatási terület egy kollaborációs keretrendszer kifejlesztése, mely megoldást nyújt a jelenlegi kollaborációs keretrendszerek hiányosságaira. Beleértve a hozzáférés kezelést, a konfliktuskezelést és a zárak kezelését mind online, mind offline változatban. A tervezett keretrendszer architektúrájának vázlata itt látható.

tracker

commit

chekout

Workspace

Indexer

Repo

Tool

Collab

Kliens

Collab

Szerver

1. ábra: Offline kollaboráció

Indexer

DB

Browser

Collab

Kliens

Collab

Szerver

Webapp

2. ábra: Online kollaboráció

Én a szakdolgozat keretén belül az offline résszel foglalkoztam, ezért csak ezt a rendszert részletezem. A modell fájlok verzióinak tárolására egy hagyományos verziókezelőt használunk, mint például az SVN vagy GIT. A repository-ban levő modellek hatékony lekérdezéséről az indexer gondoskodik, mely egy részt egy Hawk modell leképezési sémából áll, melyet a York-i egyetemen fejlesztettek, másrészt a MONDIX nevű indexerből, melyet a BME-n fejlesztettek. A modell elemeket a Cloud-ból a szerver oldali kollaborációs rendszeren keresztül lehet elérni, a kliens oldali rendszer segítségével. A kliens oldalon a workspace és az editor eszközök találhatóak. Az editorban lehet végrehajtani a modell transzformációkat, melyeket végeredménye a workspace-en jelenik meg. Az editorban végrehajtott minden változást egy workspacetracker nevű rendszer rögzíti. A helyi repository-ban levő fájlokat a hagyományos verziókezelők műveleteivel lehet rögzíteni a távoli tárolóban (chekout, commit), A feladatom az offline keretrendszer két elemének elkészítése. Egyrészt megalkotnom egy interface-t a szerver oldali kollaborációs rendszerre, másrész megtervezni, és megvalósítani a workspacetracker-t.

## Az offline rendszer műveletei

A legfontosabb műveletekre melyeket egy offline kollaborációs rendszernek el kell végeznie, interface-t készítettem. A műveletek a következők:

### Checkout

A chekout művelet lényeg, hogy visszaadja a kívánt modellt, illetve modell elemeket. A lekérdezéshez meg kell adni a helyes bejelentkezési adatokat, a kívánt verziót és a kívánt modell azonosítóját. Igény szerint meg lehet adni egy lekérdezést, ami meghatározza, hogy melyik modell elemeket akarjuk lekérdezni. A függvény visszaadja a kért modellelemeket, melyekhez van a felhasználónak olvasási jogosultsága

**public** T checkoutModel(ILoginData loginData, IVersion version,

IModel selectedModel);

**public** T checkoutModel(ILoginData loginData, IVersion version,

IModel selectedModel, Query query);

### Merge

A merge művelet lényege, hogy automatikus konfliktuskezelést hajtson végre a modelleken. A művelet egy „Three-way merge” mely a [3. ábra] –n tekinthető meg. Az R elem a központi tárolóban levő gyökér elem, mely a verziókezelő rendszerben elérhető. Az A és a B modellek, a R-ből jöttek létre különböző tranzakciók segítségével. A modell már a fel van rakva a VCS-be. A B modell most lett létrehozva. A cél egy modell létrehozása, mely tartalmazza a mindkét modellen végrehajtott változásokat, illetve, ha ez nem lehetséges, létrehoz javaslatokat az összefésülésre.

**public** T merge(T rootModel, T modelA, T modelB);

### Update

A művelet nem tartalmaz új műveletet. Egymás után végrehajt egy Checkout műveletet, és egy merge-öt. A művelethez szükséges megadni a bejelentkezési adatokat, valamint rendelkezni kell a megfelelő jogosultságokkal a modell olvasására és írására.

**public** T update(ILoginData loginData, IVersion version, String query,

T rootModel, T modelA, T modelB);

### Commit

A modellváltozások elmentésére szolgáló művelet. Bejelentkezés után, a megadott üzenettel elmenti a távoli adattárba a módosított modellt, amennyiben a helyi rendszerben elérhető az utolsó verzió, és nincs konfliktus. A művelet csak akkor hajtható végre, ha a kívánt modellelemekre a felhasználónak van írási jogosultsága.

**public** **int** commit(ILoginData loginData, T editedModeol, String commitMessage);

### Zár hozzáadása

Zárat helyez el a kíván elemre lekérdezés segítségével. Lekérdezés segítségével kombinálni lehet a durva fájl szintű zárakat, melyeket a hagyományos VCS rendszerek nyújtanak, és az elemek felsorolásával létrehozható zárakat, melyeket a modell tárolók használnak.

**public** **int** lock(ILoginData loginData, ILockQuery lockQuery);

### Zár levétele

Amennyiben a felhasználónak van jogosultsága, eltávolítja a megadott azonosítójú zárat.

**public** **int** unlock(ILoginData loginData, **int** lockID);

### Zárak listázása

Amennyiben a felhasználónak van jogosultsága, lekérdezheti a aktív szárak listáját.

**public** Set<Integer> listLock(ILoginData loginData);

### Napló

A rendszer naplóban tárolja a commitokat, valamint a commitáló felhasználó nevét, és a commithoz tartozó üzenetet. A művelet segítégével ezek listáját lehet elérni dátum szerint rendezve.

**public** Set<String> log();

### Jogosultság hozzáadása

A jogosultságok kezelése a záraknál bevezetett rendszer segítségével működik, így nem csak fájl szinten lehet szabályozni a jogosultságot. A művelet segítségével az arra jogosult felhasználó olvasási és írási jogosultságot tud adni a modell részeire a kívánt felhasználóknak, illetve felhasználói csoportoknak.

**public** **int** setPermission(String loginData, IPermissionRule permissionRule);

### Jogosultság elvétel

A josgosultságot az azonosítója segítségével ki lehet venni a rendszerből, amennyiben van ehhez jogosultsága a felhasználónak.

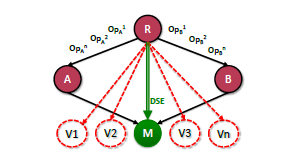
**public** **int** removePermission(String loginData, **int** permissionID);

### Jogosultságok listázása

Ha felhasználónak van hozzáférése a jogosultság kezelő eléréséhez, lekérdezheti a jelenlegi jogosultságok azonosítóit.

**public** Set<Integer> listPermissions(String loginData);

Az interfészt generikusen készítettem el, hogy többféle modellkezelést meg tudjon valósítani. Elkészítettem 3 féle specializációt is hozzá, egy interface-t amely EMF modellekhez használha, és a modelleket EMF –es resource-okban tárolja, egyet ami az eclipse specifikus IFile-ként kezeli a model fájlokat, és egy általános Java File típust kezelő interface-t. Mivel a megvalósított műveletek kivitelezése a szakdolgozat témámon kívül esik, ezért a bemutatáshoz egy skeleton programot készítettem ,a mi a műveletek végrehajtását az alapértelmezett kimentere írja ki.

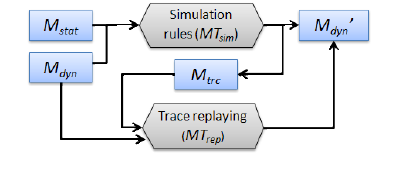


3. ábra: Three-way merge

# Fejezet

Nyomonkövetés

Biztonságkritikus rendszerekben egy fontos feladat a kód nyomonkövethetősége, azaz annak biztosítása, hogy minden módosítás után meg lehessen állapítani, hogy az adott kódrész mikor és hol változott. Ezt a hagyományos rendszerek verziókezeléssel valósítják meg, ahol a különböző verziók tárolják a kód különböző változatát. Modelvezértelt szoftverfejlesztésben nyomonkövetési modellen egy olyan modellt értünk, mely tárolja a bemeneti modellen végrehajtott transzformációkat, oly módon, hogy a nyomonkövetési modell és az eredeti modell segítségével vissza lehessen vezetni az összes változtatást a modellen. A 4. ábra szemlélteti a nyomonkövethetőségi modell szerepét. Az ábrán a négyszögek jelölik a modelleket, a hatszögek pedig a modelleken értelmezett transzformációkat. Az Mstat modell az eredeti tárolt modellt jelöli, az Mdyn modell pedig az editorba betöltött modellt, melyen a változások megtrténtek. Az Mtrc modell maga a nyomontnövethetőségi vagyis trace modell, mely a transzformációs lépések hatására létrejön. Az ábrán látható, hogy az Mdyn’ modell, mely a transzformációk eredményeképpen létrejövő modell, elérhető a transzformációk által, valamint leképezhető a nyomonkövethetőségi modell segítségével is.



4. ábra: Trace model

## Nyomonkövetés megvalósítása

A nyomonkövethetőség megvalósításához először el kellett döntenem, hogy milyen módon értesüljön a rendszere a modellen végrehajtott módosításokon. A tervezéskor két megoldás került szóba, az TransactionalEditing domain, mely a modell transzformációs lépésekkor generál értesítéseket, melyek még a modellen való végrehajtás előtt jönnek létre, illetve az Notification API használata, mely a modelleken való módosítások után hoz létre új eseményt, amelyre fel lehet iratkozni. Az előbbi megoldásban a végrehajtáskor létrejövő hibák problémát okozhatnak, mivel a valós végrehajtás előtt már a rendszer értesíti a programot, ezért a második, azaz a Notification API-t használtam.

Az API használata az EMF –es kódgenerálásban is támogatva van, azaz a generált kódban, minden művelet végrehajtásakor meghívásra kerül az eNotify(Notification) metódus, mely a modell objektumra beregisztrált összes adapter notifyChanged(Notification) metódusát meghívja.

Tehát egy EObject leszármazott osztály egy attribútumának beállításához a következő sorokat hozza létre a kódgenerátor.

**public** **void** setNewValue(Object newNewValue) {

Object oldNewValue = newValue;

newValue = newNewValue;

**if** (eNotificationRequired())

eNotify(**new** ENotificationImpl(**this**, Notification.*SET*,

OperationtracemodelPackage.*UPDATE\_ATTRIBUTE\_\_NEW\_VALUE*

oldNewValue, newValue));

}

Az eNotify művelet pedig a következő sorokból áll az BasicNotifierImpl osztályban, mely minden EObject-nek az őse.

**public** **void** eNotify(Notification notification)

{

Adapter[] eAdapters = eBasicAdapterArray();

**if** (eAdapters != **null** && eDeliver())

{

**for** (**int** i = 0, size = eAdapters.length; i < size; ++i)

{

eAdapters[i].notifyChanged(notification);

}

}

}

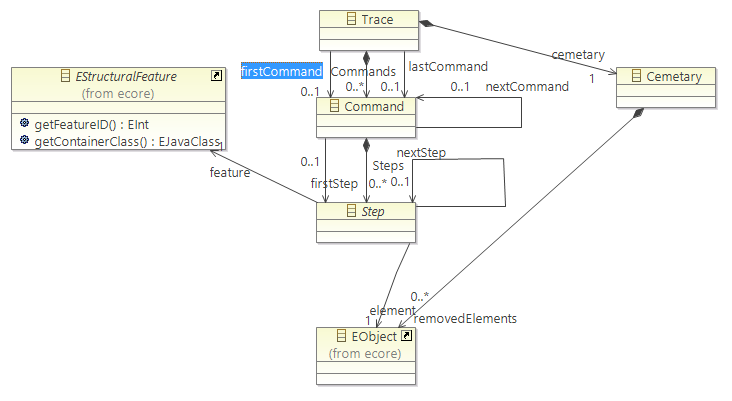
## Nyomonkövetési metamodell

Az egyes modelleken végzett tranzakciók példánymodellekben vannak tárolva, melynek szerkezetét az OperationTraceModel nevű metamodell írja le. A modell elkészítésénél ügyelni kellett rá, hogy a rendszer áttekinthető legyen, minden modellen felhasználható, valamint, hogy ha kell, a rendszer további műveletekkel bővíteni lehessen, illetve egyéb adatokat lehessen tárolni.

### A rendszer fő elemei

A rendszert az áttekinthetőség kedvéért több lépésben mutatom be. A 6. ábra a rendszer főbb összetevőit ábrázolja.

A Trace a modell gyökér eleme, mely tartalmazza a Cemetery-t, azaz a temetőt, melybe azok az elemek kerülnek, amelyek a nyomon követett modellből eltávolításra kerültek. Szintén a gyökérelem tartalmazza a listát a Commandokról, valamint a könnyebb kezelhetőség érdekében referenciákat az első és az utolsó Commandról. A rendszer úgy lett tervezve, hogy a nyomonkövetési modell mindig a követett példánnyal együtt legyen elmentve, és a két mentés közötti műveletek össze legyenek fogva. Ennek a megvalósítására lettek bevezetve a Commandok. Tehát egy Command a két mentett állapot közötti összes lépést tartalmazza. A Commandok tehát tartalmazzák a Step listákat, melyek az egyes műveleteket reprezentálják, valamint referenciákat, egyrészt a következő Command, másrészt az első Step azonosítására. A Step egy absztrakt ősosztály, mely minden, a későbbiekben bemutatott modell transzformációs lépésnek az őse. Minden transzformációs lépés azonosítja a következő lépést, valamint tartalmaz két további referenciát, az element-et, és a feature-t. Az element referencia azonosítja azt az EObject elemet, melyen a transzformáció végrehajtódott. A feature referencia EStructuralFeature típusú, és arra szolgál, hogy meghatározza, hogy az elem mely referenciáján, illetve attribútumán jött létre a változás. Ebből a két referenciából látható, hogy a rendszer direkt referenciákat használ az elemek azonosítására, aminek következtében függ a nyomon követett modelltől, vagyis a követett modellek a memóriában kell lennie mind a készítéskor, mind a felhasználáskor. Ez nagyobb modellek esetén komolyabb erőforrás igényeket vet fel, de a gyakorlati használat esetén, az eredeti modellre amúgy is szükség van.



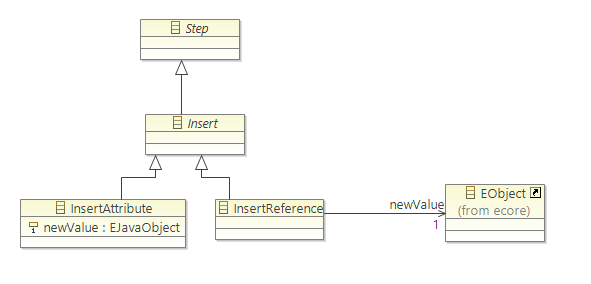
6. ábra: Nyomonkövetési metamodell

### A megvalósított műveletek

A következőkben bemutatom az általam megvalósított alapvető tranzakciós műveleteket. Minden tranzakciónál fontos megkülönböztetni, hogy azt attribútumon, vagy referencián hajtották végre, tehát minden tranzakciónak két alosztálya van, ezek reprezentálására. A referenciát megvalósító elem minden esetben EObject típusú elemre tartalmaz referenciát, míg az attribútumok változása esetén a szükséges adatokat az osztály Java Object típusú attribútumában tárolom, mivel az mindenképpen sorosítható.

#### Insert

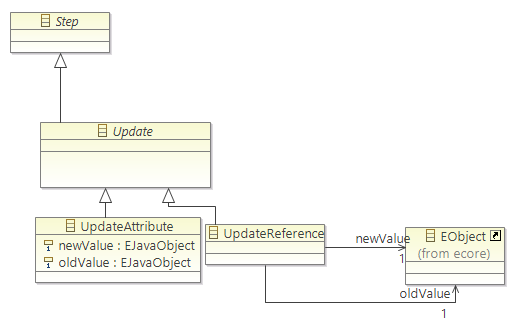
Az Insert művelet azokat a tranzakciókat tartalmazza, melyek a modellben új érték létrehozását eredményezik, tehát csak az új értéket van szükség eltárolni. A felépítését a 7. ábra mutatja.



7. ábra: Insert step

#### Update

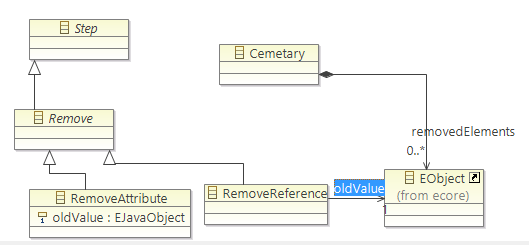
Az update műveletet akkor hozza létre a rendszer, amikor egy érték megváltozik oly módon, hogy sem a kezdeti, sem a vég állapota nem null érték. Hogy a pontos átmenetet rögzíteni lehessen, az osztály mind a régi, mind az új értéket rögzíti. Felépítését a 8. ábra ábrázolja.



8. ábra: Update step

#### Remove

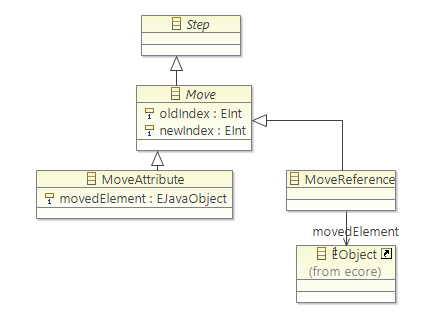
A remove művelet azokat transzformációkat tartalmazza, melynek hatására a referencia vagy az attribútum értéke null értéket vesz fel. A remove művelet miatt van szükség a rendszerben, a bár fentebb bemutatott eszközre a Cemetery-re. Az eltávolított referencia esetén a követett modellben a hivatkozás megszűnik az elemre, ezért szükséges, hogy a temetőben létrehozzunk egy új tartalmazás típusú referenciát, mely erre az elemre mutat. A 9. ábrán jól látható, hogy a RemoveReference is ugyanúgy működik, mint az előbbiekben bemutatott két művelet. Ez is csak EObject referenciát tartalmaz a régi értékre. A lényeg ott van, hogy a EObject tartalmazása kerül át az eredeti modellből a nyomonkövetési modellbe.



9. ábra: Remove step

#### Move

Egynél nagyobb multiplicitású referenciák és attribútumok esetén értelmezett a move művelet, amikor egy elem indexe megváltozik. Valójában egy áthelyezés a listában több elemnek az indexváltoztatásával jár, de a nyomonkövethetőség érdekében elég eltárolni az eredeti áthelyezést, és ennek régi és új indexét, mivel ebből már előállítható a többi index eltolódása is. A művelet az előzőekben bemutatott elven működik felépítése a 10. ábrán látható



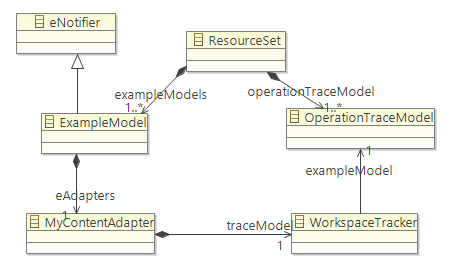
10. ábra: Move step

# Fejezet

Workspace tracker

## A workspace tracker architektúrája

A nyomonkövetési modell létrehozására és kezelésére létrehoztam a Workspace tracker nevű rendszert. A Workspace tracker architektúráját az 5. ábra mutatja be. A rendszer fő alkotó elemei a MyContentAdapter, mely az eredeti modellen levő módosításokat érzékeli, a WorkspaceTracker osztály, mely a nyomonkövetési példánymodellt építi, valamint az OperationTraceModel, mely az ecore nyelven megírt nyomonkövetési metamodell, melyet a 4.2. fejezetben mutattam be, valamint az ehhez generált java osztályok. Az ábrán látható további elemek, a ResourceSet, általában a követett modell és a trace modell egy ResourceSetbe kerül. Az ExampleModelnek a nyomon követett példánymodellt nevezem, ebből a ResourceSetben lehet egy vagy akár több is, valamint a nyomonkövetési modellből is lehet minden resourcehoz külön, illetve tudja őket egyszerre is kezelni. A továbbiakban az általam megírt három alkotóelemet mutatom be.



5. ábra: Workspace tracker architektúra

## nyomonkövetési modell elészítése

Az architektúrán látszik, hogy az OperationTraceModel-hez az adapter a WorkspaceTracker nevű java osztályon keresztül fér hozzá. Ez az osztály a felelős, a nyomonkövetési példánymodell elkészítéséért. A példánymodell építését az OperationTraceModel –ból generált java osztályokon keresztül valósítottam meg.

**public** **void** removeStep(EObject element, EStructuralFeature feature, Object

oldValue, **boolean** isAttribute) {

**if**(isAttribute){

newStep(element, feature, 5);

((RemoveAttribute)step).setOldValue(oldValue);

} **else** {

newStep(element, feature, 6);

**if**(((EReference)feature).isContainment()){

trace.getCemetary().getRemovedElements().add((EObject)oldValue);

}

((RemoveReference)step).setOldValue((EObject)oldValue);

}

}

## MyContentAdapter

A MyContentAdapter tartalmazza az WorkspaceTracker osztályt, mely a modell felépítéséért felelős. MyContentAdapter leszármaztatottja az EContentAdapter könyvtári adapternek, mely fel van készítve rá, hogy ne csak a EObjecten végrehajtott változásokat észlelje, hanem valamennyi gyermekére is.

Az adapter a különböző eseménytípusokból, az adapter felépíti az TraceModelt a generált kód segítségével.

4 eseményre van felkészítve, set, insert, remove, move

**public** **void** notifyChanged(Notification notification) {

**switch**(notification.getEventType()) {

**case** Notification.*ADD* :

**break**;

**case** Notification.*ADD\_MANY* :

**break**;

**case** Notification.*SET* :

**break**;

**case** Notification.*REMOVE* :

**break**;

**case** Notification.*REMOVE\_MANY* :

**break**;

**case** Notification.*UNSET*:

**break**;

**case** Notification.*MOVE*:

**break**;

**default**: **break**;

}

}

### Event types

#### Add

#### Set

#### Remove

#### Move

### Példánymodell elmentése

if (notification.getNotifier() instanceof Resource) {

if(!((Resource)notification.getNotifier()).isModified()){

operationTraceModel.endCommand();

}

}

# Motivációs példa

## Szélturbina

## 

# Összefoglalás

A diplomaterv összefoglalása.

# Köszönetnyilvánítás

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

# Ábrák jegyzéke

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

# Táblázatok jegyzéke

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

# Irodalomjegyzék

# Függelék

A függelék szövege.