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 nyomon követni 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

Készítette

Konzulens

2014

Tartalomjegyzék

[Összefoglaló 8](#_Toc406641557)

[Abstract 9](#_Toc406641558)

[1. Fejezet 10](#_Toc406641559)

[2. Fejezet 12](#_Toc406641560)

[2.1. Eclipse fejlesztőkörnyezet 12](#_Toc406641561)

[2.2. Eclipse Modelling Framework 12](#_Toc406641562)

[2.2.1. ECore 13](#_Toc406641563)

[2.2.2. Resource 14](#_Toc406641564)

[2.2.3. Notifikáció 14](#_Toc406641565)

[3. Fejezet 16](#_Toc406641566)

[3.1. Jelenlegi eszközök modell kollaborációra 16](#_Toc406641567)

[3.1.1. Modell adattárak 16](#_Toc406641568)

[3.1.2. Online kollaborációs rendszerek 17](#_Toc406641569)

[3.1.3. Modell verziókezelő rendszerek 17](#_Toc406641570)

[3.2. MONDO kollaborációs keretrendszer 17](#_Toc406641571)

[4. Fejezet 19](#_Toc406641572)

[4.1. Specifikáció 19](#_Toc406641573)

[4.2. Megvalósítás 21](#_Toc406641585)

[5. Fejezet 23](#_Toc406641586)

[5.1. Workspace Tracker architektúrája 23](#_Toc406641587)

[5.2. Nyomonkövethetőségi modell 24](#_Toc406641588)

[5.2.1. A metamodell fő elemei 24](#_Toc406641589)

[5.2.2. A megvalósított műveletek 26](#_Toc406641590)

[5.3. Nyomonkövethetőségi példánymodell elkészítése 29](#_Toc406641591)

[5.3.1. A modell betöltése / létrehozása 29](#_Toc406641592)

[5.3.2. Parancsok (Command) felépítése 30](#_Toc406641593)

[5.4. WTContentAdapter 31](#_Toc406641594)

[5.4.1. Értesítések kezelése 31](#_Toc406641595)

[5.4.2. A Notification műveletei 32](#_Toc406641596)

[5.4.3. Az adapter megvalósítása 32](#_Toc406641597)

[5.4.4. Esemény típusok 33](#_Toc406641598)

[5.4.5. Példánymodell elmentése 34](#_Toc406641599)

[5.5. Adapter csatlakoztatása a modellhez 34](#_Toc406641600)

[5.5.1. Kézi csatlakozás 34](#_Toc406641601)

[5.5.2. Generikus csatlakozás editorokhoz 35](#_Toc406641602)

[6. Motivációs példa 36](#_Toc406641603)

[6.1. IK4-IKERLAN 36](#_Toc406641604)

[6.2. Szélturbina vezérlő rendszer 36](#_Toc406641605)

[6.3. A rendszer működése 37](#_Toc406641606)

[7. Fejezet 41](#_Toc406641607)

[Ábrák jegyzéke 42](#_Toc406641608)

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. 17.

Vikár András

# Összefoglaló

Napjainkban egyre nagyobb hangsúlyt kap a modellalapú szoftverfejlesztés. Egyre nagyobb és komplexebb rendszerek tervezésében használják. Felmerülő probléma, hogy a modellalapú szoftverfejlesztéshez nincsenek még olyan kiforrott technológiák, mint a hagyományos kód alapú fejlesztésben. Egyik ilyen probléma az összetett kollaborációs rendszerek hiánya. Szakdolgozatom keretében egy ilyen kollaborációs rendszer tervezésébe és kivitelezésébe kapcsolódtam be.

A projektben az egyik feladatom az volt, hogy elkészítsem egy előre specifikált rendszer interfészét, mely összefogja a rendszer fontosabb műveleteit. A másik feladatom, hogy elkészítsek egy EMF (Eclipse Modeling Framework) technológiára épülő rendszert, mely a modellmanipulációs műveletek modellezi, hogy ez által a modellen végbevitt változások nyomonkövethetőek legyenek.

A dolgozat keretében elkészítettem az interfészt, valamint megterveztem és kifejlesztettem egy eszközt, mely képes az EMF technológiába tartozó modellek transzformációinak nyomonkövetésére.

# 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.

# Fejezet

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.

Összetett rendszerek esetén fontos szempont, hogy a modelleket több felhasználói környezetben is lehessen használni. A hagyományos kód alapú rendszerekben már kiforrott technológiák vannak, de modellező rendszereknél, még nincs olyan rendszer, mely minden ehhez szükséges feladatot ellát, mint a modell tárolás, a felhasználók hitelesítése, modellek összefésülése és a konfliktuskezelés. A csoportnak mellyel együtt dolgoztam az volt a célja, hogy megalkosson egy összetett rendszert, mely lehetővé teszi a modelleken való együttes munkát, mind online, mind offline környezetben.

A feladatom az volt, hogy készítsek el egy interfészt, melyen keresztül a rendszer alapvető műveletei megvalósíthatóak, valamint megtervezni és megvalósítani a Workspace Tracker nevű rendszert, mely eltárolja a modelleken végrehajtott változásokat.

A workspace tracker legfontosabb műveletei:

* modell független, azaz bármilyen modell felett működik
* áttekinthető struktúra, hogy a végrehajtott lépések könnyen visszafejthetőek legyenek
* könnyű kezelés: a nyomonkövethetőség ne okozzon plusz feladatot a fejlesztőknek.

A leírt feladatokat szakdolgozatom során véghezvittem. Ezen dokumentum célja a fejlesztőeszköz elkészítésének bemutatása a kezdeti irodalomkutatástól, a fejlesztés bemutatásán át a fejlesztett rendszer bemutatásáig. A dolgozatban bemutatom a fejlesztés alatt felhasznált technológiákat és eszközöket (2. Fejezet), majd ismertetem a jelenlegi kollaborációs rendszerek és a tervezett keretrendszer közötti különbséget, valamint ennek felépítését (3. Fejezet). A 4. Fejezetben leírom a megvalósítandó rendszer specifikációját, majd leírom az általam elkészített és megvalósított interfészt. Ezt követően ismertetem Workspace Tracker tervezését, és bemutatom a rendszer alkotóelemeit (5. Fejezet), és végül bemutatom a rendszer működését (6. Fejezet)

# Fejezet

Háttér technológiák

A feladatomat Java nyelven végeztem Eclipses technológiák segítségével.

## Eclipse fejlesztőkörnyezet

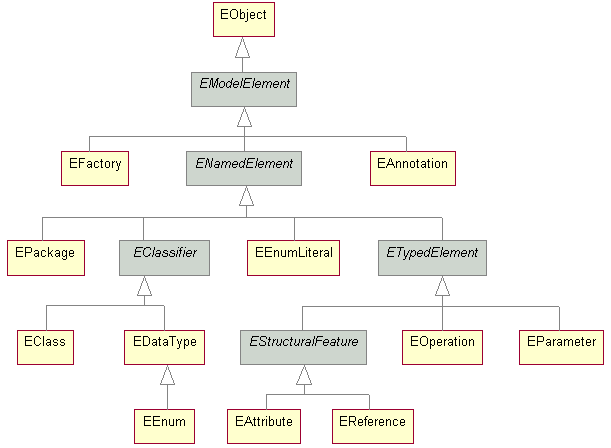
Az Eclipse[7] nyílt forráskódú, platform független szoftverkeretrendszer, mely integrált web és alkalmazásfejlesztő eszközöket tartalmaz. Maga a platform nem nyújt sok végfelhasználói funkcionalitást, ezeket az integrált plug-in-ek adják hozzá a rendszerhez

## Eclipse Modelling Framework

Napjaink egyik legelterjedtebb, modell alapú architektúra fejlesztő eszköze az Eclipse Modelling Framework (EMF)[16]. A keretrendszerrel szakterület-specifikus nyelvek metamodelljének megalkotását, és a modellből való kódgenerálást. Az eszköz számos modellező nyelvet támogat, melyek közül a legelterjedtebb az ECore[10].

### ECore

ECore nyelven metamodelleket tudunk készíteni a keretrendszer által nyújtott meta-metamodell keresztül. Ennek az elemkészlete az úgynevezett E osztályok, melyek struktúráját az 1. ábra mutatja.



1. ábra: Ecore felépítése[20]

A modellen ábrázolt fontosabb alkotóelemek a következők.

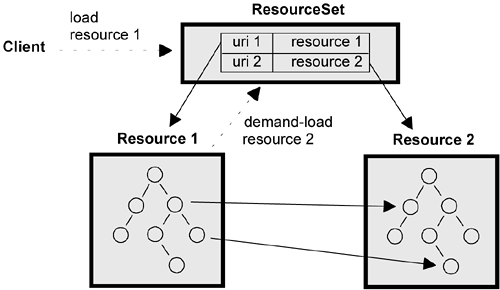
* **EObject**: Minden ECore osztály őse. Az osztály implementálja a Notifier interfészt, melynek segítségével a modellpéldányon a változások nyomonkövethetőek.
* **EClass**: Osztályok reprezentál. Egy osztály attribútumokat, referenciákat és operációkat tartalmazhat.
* **EAttribute**: Típusos attribútumot reprezentál, melynek neve és típusa van. A példánymodellben a típusnak megfelelő értéket lehet adni.
* **EReference**: 2 osztály közötti egyirányú asszociációt reprezentál. Egy referenciának van neve, számossága, valamint megadható, hogy a referencia tartalmazást valósít-e meg.

A metamodellt Ecore diagram szerkesztővel lehet létrehozni az létrehozott modellből EMG generátor modell segítségével lehet forráskódot generálni. A generátor modell az alábbi kódokat generálja ki:

* **Modell**: Legenerálja a metamodellben szereplő összes osztályt és a factory osztályokat. Minden elemhez külön interfészt és implementációs osztályt hoz létre
* **Edit**: A modell szerkesztését lehetővé tevő osztályok
* **Editor:** Létrehoz egy Eclipse plugint, melynek segítségével példánymodellek hozhatók létre grafikusan
* **Test:** Modellhez tartozó teszteket generálja

### Resource

Az EMF egy példánymodellt egy Resource-ben tárol, A Resource-ben található a fa struktúrájú példánymodell. A Resource-okat a keretrendszer ResourceSet-ben tárolja, melyben az összetartozó modellek kerülhetnek. A Resource-okat URI segítségével lehet azonosítani. A ResourceSet felépítését a 2. ábra mutatja.

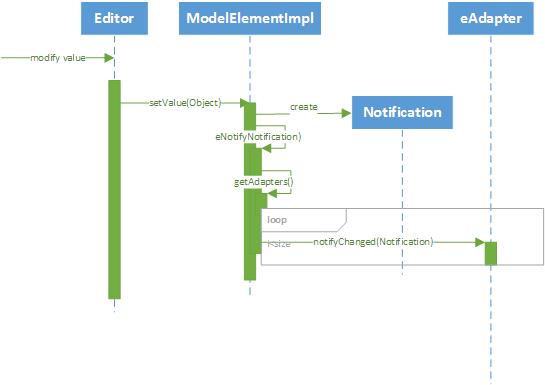


2. ábra: ResourceSet felépítése[21]

### Notifikáció

Az EMF-ben támogatva van a modelleken történt változások észlelése. Ezt a rendszer Notifier interfészen keresztül valósítja meg observer tervezési mintát használva. A ResourceSet, a Resource és az EObject is implementálja ezt az interfészt. A generált kódban, minden művelet végrehajtásakor meghívásra kerül az eNotify(Notification) metódus, mely a modell objektumra beregisztrált összes adapter notifyChanged(Notification) metódusát meghívja.

Egy attribútum beállításának folyamatát mutatja be a 3. ábra. A változtatást egy editoron keresztül hajtják végre, ez meghívja az adott modellelemet reprezentáló osztály attribútum beállító függvényét, mely az ősöktől származtatott függvények segítségével eléri az adaptereit.



3. ábra: Notifikáció

# Fejezet

Kollaborációs rendszer

A MONDO EU FP7 projekt egyik kutatási területe a kollaboráció modellező rendszerekben. A projektben kiadott cikk () alapján áttekintettem a jelenlegi megoldásokat, és megalkottam a specifikációnak megfelelő rendszer interfészét.

## Jelenlegi eszközök modell kollaborációra

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 összefésülés. Sajnos a tudomány jelenlegi állásában a modellező technológiák még nem zárkóztak fel jelenlegi eszközökhöz, sem funkciókban, sem érettségben. Ezért a fejlesztőeszköz készítőknek ad hoc megoldásokat kell megvalósítaniuk. A következőkben áttekintjük ezeket a megoldásokat.

### Modell adattárak

A modell adattárak modellezett eszközök tárolására megalkotott tároló rendszerek, melyek leginkább a perzisztenciára és a több infrastruktúrán való egyidejű hozzáférésre (kliens-szerver) összpontosítanak. Ezek az eszközök csak korlátozottan támogatják az olyan technológiákat, mint a konfliktuskezelés, az elágazások kezelése, valamint a modell összehasonlítás. Ehelyett kiterjesztett mechanizmusokat és központi API-kat nyújtanak, hogy kiegészítő, funkció specifikus eszközöket lehessen használni. Ilyen technológiák a következők:

* Eclipse Modeling Framework Connected Data Objects (EMF CDO)[9]
* Dawn Projekt[11]
* MORSA[12]

### Online kollaborációs rendszerek

Az online kollaboratív modellező rendszerek rövid tranzakciós modellekre támaszkodnak, melyben több felhasználó egyetlen modellpéldányt módosít valós időben. Ezekben a rendszerekben nincs konfliktuskezelés, vagy csak nagyon egyszerű mechanizmusokat használnak. Ennek következtében a konfliktusok nagyon korlátozott körre terjednek ki, és azonnal meg kell őket oldani. Ennek az ára a kommunikációs többletterhelés, valamint hogy minden szereplőnek egy időben online kell lennie. Ilyen eszközök például:

* CoolModes (COllaborative Open Learning and MODEling System)[13]
* EMFCollab[14]
* SpacEclipse-CGMF[4]

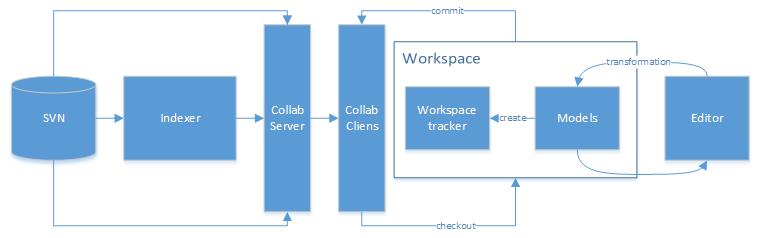
### Modell verziókezelő rendszerek

A verziókezelő rendszerek szorosabban kapcsolódnak az offline kódalapú verziókezelő rendszerekhez, mint amilyen a CVS és az SVN is. Hosszú tranzakciós modelleket használnak, melyek alapja, hogy a felhasználók nagyobb munkákat, változtatásokat tesznek közzé. Emiatt a konfliktusok gyakoriak, így a rendszerek ezek felderítésére, feloldására és összefésülésé koncentrálnak. Architektúrától függően lehet, hogy nem is foglalkoznak olyan kiegészítő funkciókkal, mint a hitelesítés, hozzáférés kezelés. Ilyen rendszerek például:

* ModelCVS[17]
* AMOR[15]
* Eclipse Modeling Team Framework[8]
* EMFStore[6]

## MONDO kollaborációs keretrendszer

A MONDO kutatás egyik célja, hogy megalkosson egy modellező rendszerekhez használható kollaborációs rendszert a jelenleg létező technológiák segítségével. A rendszer támogatja mind az online, mind az offline együttműködést, fejlett konfliktuskezeléssel és szabály alapú hozzáférés kezeléssel. A keretrendszer online és offline komponensekből épül fel. A szakdolgozatom során az offline komponensek megvalósításával foglalkoztam. Az architektúrát a 4. ábra mutatja.



4. ábra: Offline kollaboráció

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 Hawk és egy MONDIX nevű rendszerből épül fel. A Hawk-ot a Yorki egyetemen fejlesztették ki. A rendszer szövegalapú modell-fregmensekből épít fel modelleket, és ezeken végzi az indexelést. A Hawk kimeneti modelljén a MONDIX nevű rendszer, melyet a BME-n fejlesztettek ki, végez indexelést a modell lekérdezések gyorsítása érdekében. 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 a workspace tracker 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. (i) Egyrészt megalkotnom egy interfészt a szerver oldali kollaborációs rendszerhez, (ii) másrész megtervezni és megvalósítani a Workspace Tracker-t.

# Fejezet

Kollaborációs interfész

Az szerver oldali interfész feladata, hogy összefogja azokat a műveleteket, amiket a rendszernek el kell tudnia végezni.

## Specifikáció

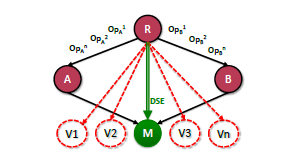
A rendszer specifikációja a D4.1 – Collaboration primitives and patterns[3] dokumentumban található meg. Az alábbi műveletek megvalósítása szükséges a rendszer működéséhez.

### Checkout

A checkout művelet lényege, hogy visszaadja a kívánt modellt, illetve a 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

### Összefésülés (Merge)

A összefésülés művelet lényege, hogy automatikus konfliktuskezelést hajtson végre a modelleken. A művelet egy „Three-way merge” mely az 5. ábrán tekinthető meg. Az R elem a helyi tárolóban levő gyökér elem, melyen a változásokat a felhasználó végrehajtotta. Az A és a B modellek, a R-ből jöttek létre különböző tranzakciók segítségével. Az A modellt már a feltöltötték a VCS-be. A B modellt most szeretnénk kommitálni. A cél egy modell létrehozása, mely tartalmazza a mindkét modellen végrehajtott változásokat, illetve, ha ez nem lehetséges, létrehoz javaslatokat az összefésülésre.



5. ábra: Three-way összefésülés

### Frissítés (Update)

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

### Kommitálás (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.

### Zár hozzáadása

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

### Zár levétele

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

### Zárak listázása

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

### Napló

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

### 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.

### Jogosultság elvétel

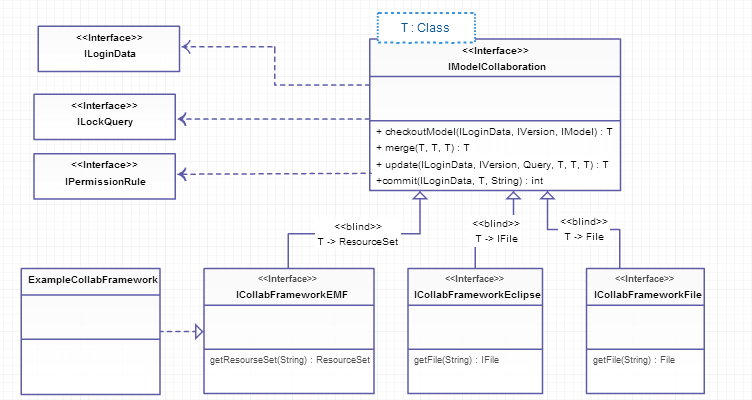
A jogosultságot az azonosítója segítségével ki lehet venni a rendszerből, amennyiben van ehhez jogosultsága a felhasználónak.

### Jogosultságok listázása

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

## Megvalósítás

Az interfészt generikusan készítettem el, hogy többféle modellkezelést meg tudjon valósítani. Elkészítettem 3 féle specializációt is hozzá, (i) egy interfészt amely EMF modellekhez használható, és a modelleket EMF –es resource-okban tárolja, (ii) egyet, ami az Eclipse specifikus IFile-ként kezeli a modell fájlokat, (iii) és egy általános Java File típust kezelő interfészt. Mivel a megvalósított műveletek kivitelezése a szakdolgozat témámon kívül esik, ezért a bemutatáshoz egy skeleton programot készítettem, ami a műveletek végrehajtását az alapértelmezett kimenetre írja ki.



6. ábra: Interfész felépítése

A 6. ábrán nem ábrázoltam az összes megvalósított függvény, melyek a következőek:

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

IModel selectedModel);

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

IModel selectedModel, Query query);

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

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

T rootModel, T modelA, T modelB);

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

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

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

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

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

**public** **int** setPermission(ILoginData loginData, IPermissionRule

permissionRule);

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

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

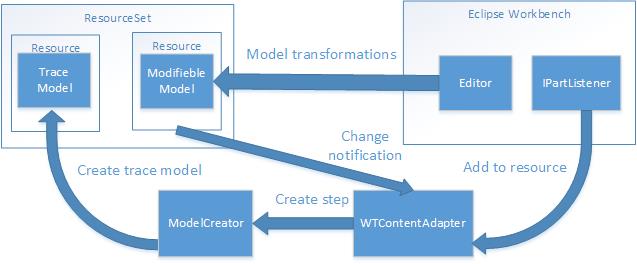
1. kód: Kollaborációs interfész

# Fejezet

Modellmanipulációs műveletek nyomonkövethetőségi modellezése

## Workspace Tracker architektúrája

Biztonságkritikus rendszerekben egy fontos feladat a kód nyomonkövethetősége[1], 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. Modell vezérelt szoftverfejlesztésben fontos, hogy az egyes verziókon kívül elérhető legyen, hogy modellen milyen műveletek segítségével hajtották végre a változást. Ehhez megvalósítottam a Workspace tracker nevű komponenst, mely egy trace modellben, azaz nyomonkövethetőségi modellben tartalmazza a transzformációkat.



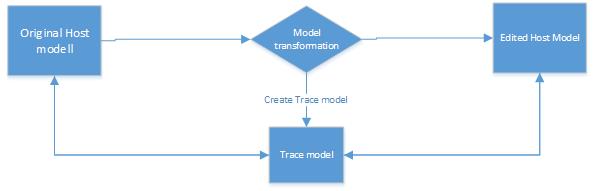
7. ábra: Workspace tracker architektúra

Workspace tracker architektúráját az 7. ábra mutatja be. A rendszer általam elkészített alkotóelemei (i) a WTContentAdapter, mely az eredeti modellen véghezvitt módosításokat a Resource által megvalósított Notifier interfészen keresztül érzékeli, majd ezeket az eseményeket, melyek tartalmazzák a változtatás adatait, leképezi a nyomonkövethetőségi modell lépéseire. (ii) A Trace Modell, mely az Ecore nyelven megírt nyomonkövethetőségi metamodell, valamint az ehhez generált java osztályok, melyek segítségével a nyomonkövethetőségi példánymodellt létrehozza az adapter. Az ábrán látszik, (iii) hogy az adapter a modellt a ModelCreatoron keresztül éri el. A megfigyelt példánymodellt és a hozzá tartozó nyomonkövethetőségi példánymodellt általában egy ResourceSet-ben tároljuk, mivel szorosan összekapcsolódnak. (iv) A rendszerhez hozzátartozik még egy komponens, mely az Eclipses EMF editorokban szerkesztett modellekhez betölti, vagy létrehozza a nyomonkövethetőségi modellt. A Funkció bekapcsolásakor egy listener a megnyitott editorokból a Workbench technológia[19] segítségével eléri a Resource-okat, melyeken dolgoznak, és az editor ResourceSet-jébe behelyezi a nyomonkövethetőségi modellt

## Nyomonkövethetőségi modell

A nyomonkövethetőségi modell feladata tehát, hogy minden modell transzformációs lépést eltároljon, oly módon, hogy be lehessen azonosítani a lépések sorrendjét, valamint, hogy melyik modell elemen lett a változás végrehajtva. Az 8. ábra mutatja, hogy a trace modell építés a műveletek végrehajtásakor történik, és segítségével az eredeti állapot visszaállítható, illetve az eredetiből a módosított állapot is.

Az egyes modelleken végzett tranzakciók példánymodellekben vannak tárolva, melynek szerkezetét az Trace modell metamodellje 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.

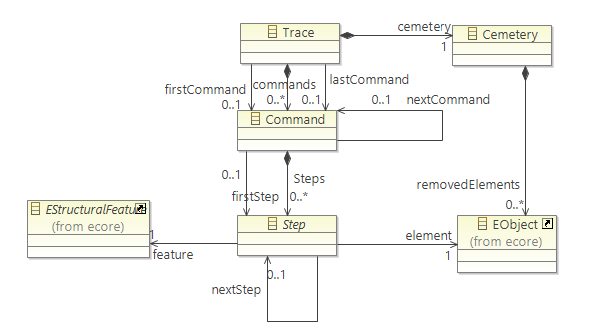


8. ábra: Modell nyomonkövethetőség

### A metamodell fő elemei

A rendszert az áttekinthetőség kedvéért több lépésben mutatom be. A 9. ábra a rendszer főbb összetevőit ábrázolja.

A Trace a modell gyökér eleme, mely tartalmazza a temetőt (Cemetery-t), azaz a temetőt, melybe azok az elemek kerülnek, amelyek a nyomon követett modellből eltávolításra kerültek. Szintén a gyökérelem tartalmazza a listát a Command-okról, valamint a könnyebb kezelhetőség érdekében referenciákat az első és az utolsó Command-ról. A rendszer úgy lett tervezve, hogy a nyomonkövethetőségi 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.



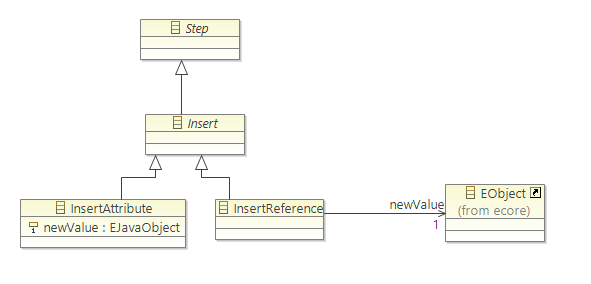
9. ábra: Nyomonkövethetőségi metamodell

### 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.

#### Beszúrás (Insert)

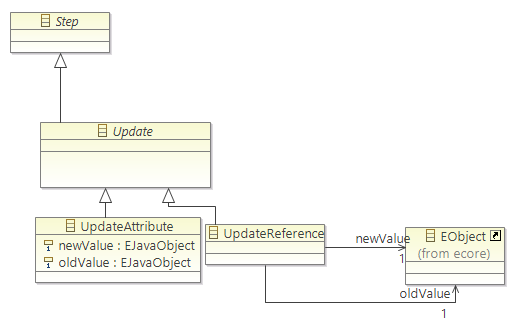
A beszúrás művelet azokat a tranzakciókat tartalmazza, melyek a modellben új érték létrehozását eredményezik, tehát csak az új értéket van szükség eltárolni. A felépítését a 10. ábra mutatja.



10. ábra: Beszúrás

#### Frissítés (Update)

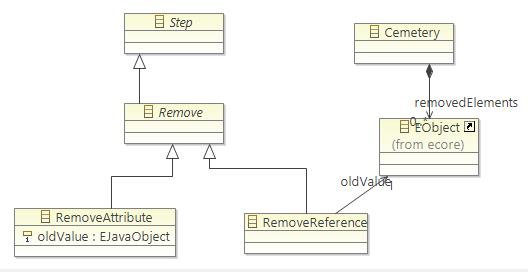
A frissítés műveletet akkor hozza létre a rendszer, amikor egy érték megváltozik oly módon, hogy sem a kezdeti, sem a vég állapota nem null érték, vagyis végig mutat valamely objektumra. Hogy a pontos átmenetet rögzíteni lehessen, az osztály mind a régi, mind az új értéket rögzíti. Felépítését a 11. ábra ábrázolja.



11. ábra: Frissítés

#### Eltávolítás (Remove)

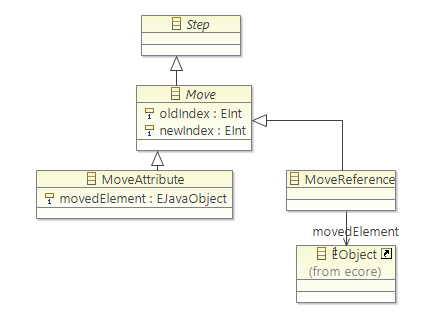
Az eltávolítás művelet azokat transzformációkat tartalmazza, melynek hatására a referencia vagy az attribútum értéke null értéket vesz fel, vagyis az attribútum, illetve a referencia nem mutat objektumra. A remove művelet miatt van szükség a rendszerben, a már fentebb bemutatott eszközre a temetőre. 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 egy EObject referenciát tartalmaz a régi értékre, melyet vagy a követett modell tartalmaz továbbra is, vagy bekerül a temetőbe.



12. ábra: Eltávolítás

#### Áthelyezés (Move)

Egynél nagyobb multiplicitású referenciák és attribútumok esetén értelmezett az áthelyezés 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



13. ábra: Áthelyezés

## Nyomonkövethetőségi példánymodell elkészítése

Az architektúrán látszik, hogy az Trace modellhez az adapter a ModelCreator nevű java osztályon keresztül fér hozzá. Ez az osztály a felelős, a nyomonkövethetőségi példánymodell elkészítéséért. A példánymodell építését az OperationTraceModel-ből generált java osztályokon keresztül valósítottam meg.

### A modell betöltése / létrehozása

A ModelCreator konstruktorában kell megadni a ResurseSet-et és az URI-t melyek megmondják, hogy hol legyen tárolva a nyomonkövethetőségi modell, valamint megadható hogy a trace modell létezik-e vagy most akarunk újat létrehozni.

Új modell esetén létrehozza a megadott URI-val azonosított resource-t és létrehozza az alapelemeket, a Trace-t és a temetőt.

Létező nyomonkövethetőségi modell esetén meg kell keresni az utolsó Command-ot, és ennek az utolsó Step-jét, hogy a trace motelt folytatni lehessen. Ezt mutatja a következő kódsor.

**if** (traceModel.getContents().get(0) **instanceof** Trace) {

traceModel = resourceSet.getResource(uri, **true**);

trace = (Trace)traceModel.getContents().get(0);

command = ((Trace)traceModel.getContents().get(0)).getLastCommand();

step = command.getSteps().get(0);

**while** (step.getNextStep() != **null**) {

step = step.getNextStep();

}

}

2. kód: Nyomonkövethetőségi modell betöltése

### Parancsok (Command) felépítése

A modell építését lényegében két dolog határozza meg. Kell-e új parancs, és milyen típusú a beszúrandó lépés (Step). Ezért ezt az öt publikus függvényt valósítottam meg:

**public** **void** insertStep(EObject element, EStructuralFeature feature, Object newValue, **boolean** isAttribute);

**public** **void** updateStep(EObject element, EStructuralFeature feature, Object oldValue, Object newValue, **boolean** isAttribute);

**public** **void** removeStep(EObject element, EStructuralFeature feature, Object oldValue, **boolean** isAttribute);

**public** **void** moveStep(EObject element, EStructuralFeature feature, Object movedElement, **int** oldIndex, **int** newIndex, **boolean** isAttribute);

**public** **void** endCommand();

3. kód: Modell építő metódusok

A különböző lépést létrehozó függvények működés nagyon hasonló. Először létrehoznak egy típusuknak megfelelő elemet, majd az element és a feature paraméterek továbbadásával meghívják a newStep privát metódust, mely ezen adatok alapján a Step ősosztály két referenciáját beállítja, valamint új parancsot hoz létre, ha erre szükség van. Valamint beállítja a Command és a Step referenciáit, a következő lépésekre. Ezek után pedig az osztály specifikus értékeket beállítják. Példának a removeStep metódus kódját mutatom, be mely a legbonyolultabb, köszönhetően a temető kezelésének. Látható, hogy a függvény alapvetően két részre oszlik attól függően, hogy a változtatott paraméter referencia vagy paraméter. Ezen kívül referencia esetén meg kell vizsgálni, hogy a törölt adat a szülővel tartalmazási viszonyban volt-e, amennyiben nem volt, nem kell mást csinálni, mint megadni a régi referencia értékét, viszont, ha tartalmazás ált fenn, akkor a törlés után, az eredeti modellben nem lesz megtalálható a kívánt EObject, ezért át kell helyezni a temetőbe, mely tartalmazás típusú referenciában tárol EObject-eket.

**public** **void** removeStep(EObject element, EStructuralFeature feature, Object oldValue, **boolean** isAttribute) {

**if**(isAttribute){

Step step = OperationtracemodelFactory.*eINSTANCE*

.createRemoveAttribute();

newStep(element, feature, step);

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

} **else** {

Step step = OperationtracemodelFactory.*eINSTANCE*

.createRemoveReference();

newStep(element, feature, step);

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

trace.getCemetery().getRemovedElements()

.add((EObject)oldValue);

}

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

}

}

4. kód: Eltávolítás lépés megvalósítása

Az endCommand() metódus egy nagyon egyszerű metódus, ami nem csinál mást, mint a jelenlegi command-ot beállítja a Trace lastCommand referenciájába. Ugyanis a lastCommand reprezentálja az utolsó befejezett command-ot, mint egy végrehajtott tranzakciót. A newStep metódus is ezt kihasználva nézi meg, hogy szükség van-e új command létrehozására, amennyiben a jelenlegi command megegyezik az utolsó command-dal a lépés beszúrásának idején, akkor előtte már véget kellett, hogy érjen az előző parancs, és ilyenkor az új lépést, már egy új commandba kell helyezni.

## WTContentAdapter

### Értesítések kezelése

A nyomonkövethetőség megvalósításához először el kellett döntenem, hogy milyen módon értesüljön a rendszere a modellen végrehajtott módosításokról. A tervezéskor két megoldás került szóba, az TransactionalEditing domain[18], 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.

### A Notification műveletei

A technológia arra épül, hogy a változások adatait a Notification osztályban eltárolja. Az osztályból minden olyan adat elérhető, amely a nyomonkövethetőségi modell elkészítéséhez kell. A Step osztály element referenciájának az értéke tartalmazza az objektumot, amelyen a változás történt. Ez az objektum a getNotifier() műveleten keresztül érhető el. A feature referencia, melyben azt tároljuk, hogy mely referencián vagy attribútum értékén történt a változás a getFeature() metódus segítségével kap értéket. A régi és az új értékeket a getOldValue() és a getNewValue() függvények adják meg. A metamodellből generált kód, minden modellmódosító függvénye átad egy ilyen típusú objektumot az összes regisztrált adapterének.

### Az adapter megvalósítása

Az események észleléséhez nem kellett mást csinálni, mint elkészíteni az adapter, mely a transzformációs eseményekből a Notification adatait felhasználva létrehozza a nyomonkövethetőségi modell lépéseit. Ez az adapter a WTContentAdapter.

A WTContentAdapter tartalmazza az ModelCreator osztályt, melyen keresztül felépíti a trace példánymodellt. WTContentAdapter leszármaztatottja az EContentAdapter könyvtári adapternek, mely fel van készítve rá, hogy amikor egy Notifier-t implementáló osztály adapterei közé bekerül, akkor ezzel együtt bekerül ennek az összes gyermekobjektumába is.

Az eseménykezelés megvalósításához felül kell írni az adapter notifyChange(Notification) metódusát. Az objektumon végbevitt változások különböző esemény típusokat generálnak. Ezeket az eseményeket kellett leképezni, a nyomonkövethetőségi modell különböző Step alosztályaira. Az események kezelésére a következő switch case szerkezetet használtam.

**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**;

}

}

5. kód: Adapter notifyChanged metódusának felépítése

### Esemény típusok

A Notification osztály több eseményre van felkészítve. Ezek közül a legfontosabbak: Add, Add many, Set, Unset, Remove, Move. Az adapter feladata tehát, hogy a bekövetkezett eseményeket leképezze a nyomonkövethetőségi modell különböző lépéseire, melyeket az 5.2. fejezetben fejtettem ki. Ehhez meg kell hívni a 3. kódban deklarált függvények valamelyikét. Az EMF példánymodell multiplicitás szerint csoportosítja a műveleteket. Egynél nagyobb multiplicitású tulajdonság esetén beszúráskor az „Add” vagy „Add many”, kivételkor pedig a „Remove” vagy a „Remove many” esemény jön létre, egy multiplicitású adat beállításakor pedig „Set”, illetve „Unset” esemény. Az nyomonkövethetőségi modell megalkotásakor én nem multiplicitás szerint bontottam a műveleteket. Az én megközelítésemben a lényeg azon van, hogy elérető-e az új illetve a régi érték. listába való beszúráskor nem tudunk régi értékről beszélni, ezért az ilyen események bekövetkeztekor mindig „Insert step” jön létre, ugyanígy a listából való töltéskor mindig csak a régi értéknek van értelme, ezért ilyenkor mindig „remove step” jön létre.

Set esemény létrejöttekor három lehetőség van. Amikor a régi érték null érték, akkor „insert step” jön létre, amikor az új érték a null, akkor „remove step”, amikor pedig egyik sem null értékű, tehát az érték átírása történt, akkor „update step jön létre”. Az előzőektől lényegesen különbözik a „move” esemény, mely a listában történő index változásokra figyel. Ennek egyértelmű megfeleltetése a nyomonkövethetőségi modell „move step” eleme.

### Példánymodell elmentése

A feladat egy fontos része volt, hogy észlelni kell az eredeti modell mentését. Két mentett állapot közötti módosítás lépései kerülnek egy Command-ba.Ezenkívül arra is figyelni kellett, hogy a trace modell csak akkor legyen elmentve, amikor az eredeti modell, mivel a kereszthivatkozások miatt keletkezhetnek olyan elemek, melyek a mentetlen eredeti modellben nem léteznek. A Resource sorosításakor az adapter értesül az eseményről. A Resource isModeified() metódusa addig tér vissza hamis értékkel, amíg mentés, illetve betöltés esetén nem történt változás a resource szerkezetében. A mentéskor létrejövő eseménykor tehát az értéke biztosan hamis. Minden más esetben pedig igaz lesz.

if (notification.getNotifier() instanceof Resource) {

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

operationTraceModel.endCommand();

this.getTraceModel().save(Collections.EMPTY\_MAP);

}

}

6. kód: Nyomonkövethetőségi modell mentése

## Adapter csatlakoztatása a modellhez

A rendszer működéséhez a modellhez csatlakoztatni kell az elkészített WTContentAdapter-t. Alap esetben a Resource-hoz érdemes csatlakoztatni az adaptert. Ilyenkor a modell minden elemének módosulására figyel a rendszer, de igény szerint a modell bármely eleméhez hozzá lehet csatlakoztatni, de ebben az esetben csak az a modellelem, és annak gyermekelemeire figyel a rendszer. Az adapter csatlakoztatásához létrehoztam egy plugin menüt, de lehetőség van kézi csatlakozásra is.

### Kézi csatlakozás

Amennyiben a programozó hozzáfér az editor kódjához, illetve, ha a modellt nem editor, hanem kód segítségével hozza létre, akkor kézzel tudja hozzáadni az adaptert a példánymodellhez. Ehhez a Notifier interfészt megvalósító valamely osztály (ResoursSet, Resource, EObject) adapterei közé kell felvenni egy WTContentAdapter példányt. A WTContentAdapter példányosításakor meg kell adni a ResourceSet-et, és az URI-t a trace modell tárolásához, valamint hogy létezik-e már a trace modell. A példakód hozzáadja az adaptert a modellt tartalmazó resource-hoz, és létrehoz egy új trace modellt a megadott azonosítóval a modellt tartalmazó ResourceSet-be.

resource.eAdapters().add(**new** WTContentAdapter(resource.getResourceSet(),

URI.*createURI*(path, **true**), **false**));

7. kód: Adapter kézi csatolása

### Generikus csatlakozás editorokhoz

Általában a modellt kezelő felhasználó a modellt editoron keresztül éri el, melynek forráskódjához nem fér hozzá, ezért nem tudja a kézi módszerrel beregisztrálni az adaptert. Ennek megoldására létrehoztam egy Eclipse plugint, mely létrehoz egy menüt, és ebben egy állapottal rendelkező parancsot. A menüelemet kiválasztva a rendszer a megnyitott editorban, és minden új megnyíló EMF editor modelljéhez beregisztrálja a WTContentAdapter-t, mely létrehozza, illetve, ha már létezik, akkor betölti a hozzá tartozó nyomonkövethetőségi modellt.

Ahhoz, hogy az adaptert hozzá lehessen adni a modellhez, hozzá kell férni a Resource-hoz, melyben az editor az adatait tárolja. Ezt minden olyan editorból meg lehet szerezni, mely megvalósítja az IEditingDomainProvider interfészt. Az újonnan megnyitott ablakok érzékelésére létrehoztam egy IPartListenert, melyet a parancs kiválasztásakor az aktív WorkbenchPage-re beregisztrálok, a parancs kikapcsolásakor leregisztrálom. Szintén a WorkbenchPage-en keresztül el lehet érni az összes megnyitott editort, melynek segítségével a megnyitott modellekhez is létrejön a trace modell. Minden nyomonkövethetőségi modell nevét az eredeti modell nevéből származtatok, így garantálva az egyediséget.

# Motivációs példa

## IK4-IKERLAN

IK4-IKERLAN egy spanyol technológiai központ, melynek középpontjában az innováció és teljeskörű termékfejlesztés van. A cégnek 40 éves tapasztalata van a mechanika, az elektronika és a számítástechnika ötvözésében, az iparban. Fejlett technológiaátadásokat végeztek a legkülönbözőbb területeken, beleértve a közlekedést (vasúti és felvonók), az energiát (szél- és napenergia, és tároló rendszerek), automatizálást, és még sok mást

IK4-IKERLAN már több mint 10 éve dolgozik szélturbinák felügyeleti és vezérlő rendszereivel. A világ egyik vezető vállalata a megújuló energiaforrások területén. Több mint 6 éve használják a Model Driven Engineering (MDE) paradigmát a felügyeleti és vezérlőrendszerek tervezésében.

Az MDE segítségével alkottak meg egy szélturbina szakterület-specifikus modellező eszközt, amelynek elsődleges célja a termelékenység növelése szélturbina rendszerek fejlesztésében.

Az IK4-IKERLAN célja az, hogy kiterjesszék MBE technológiát más tevékenységekre is, mint a szélturbina vezérlés szabályozása, az ügyfelek egyedi igényei szerint. Ez ahhoz vezethet, hogy a felhasználók száma a modellező folyamatokban megnő. Jelenleg azonban a felhasználók nem tudnak hatékonyan elosztottan dolgozni a rendszeren. Erre a problémára keres megoldást a MONDO projektben.

## Szélturbina vezérlő rendszer

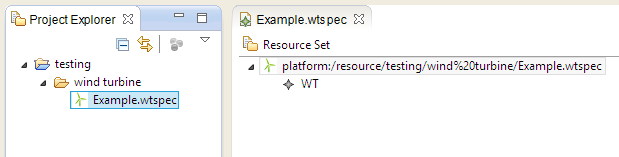
MBE segítségével fejlesztettek ki egy szélturbina vezérlő rendszert[2]. Szélturbina Domain-specifikus modellezési eszközök segítségével modellezik a vezérlő alkalmazást.

Egy adott modell konfigurációban egy szélturbina közel 2000 alapvető vezérlőegységből állhat, melyek mintegy 2000 bemenet és akár 1000 kimenetek is tartalmazhatnak. A vezérlőegység alapvető és újrafelhasználható szabályozó elem, amely kombinálható más vezérlőegységekkel, hogy létrejöjjön egy komplexebb elem. Ezeket nevezzük alrendszereknek. A modell gyökér eleme a Szélturbina WT (Wind turbine). Ebben lehet tárolni a rendszer alkotó elemeit: az alrendszereket, a bemeneteket, kimeneteket, paramétereket, változókat, hibákat, melyeket a vezérlőegységek használnak. A vezérlőegységeket csak alrendszerhez lehet felvenni.

A modellező eszköz Ecore metamodelljét megkaptam, és ebből legeneráltam a szükséges kódokat (Model, Edit, Editor). Az elkészített rendszer működését ezen az EMF által generált editoron keresztül mutatom be.

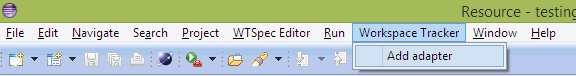
## A rendszer működése

A rendszer teszteléséhez el kell indítani egy runtime eclipse-t, majd létre kell hozni egy új wtspec példánymodell WT gyökérelemmel.



14. ábra: Szélturbina példánymodell létrehozás

Az EMF editor megjeleníti a teljes ResourceSet tartalmát. A 14. ábrán látható, hogy jelenleg nincs más modell a halmazban, tehát a nyomonkövethetőségi modell még nem lett létrehozva. Az adapter hozzáadásához a menüsoron kell kiválasztani a Workspace Tracker menüt, azon belül pedig az Add adapter parancsra kell kattintani, amennyiben még nincs mellette pipa.

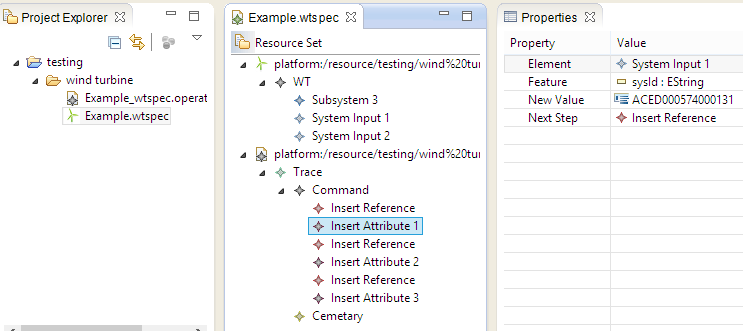


15. ábra: Adapter hozzáadás

A kattintás után a megnyitott editorban azonnal fel kell tűnnie egy új operationtracemodel típusú modellnek, melynek trace gyökérelemében egyetlen temető elem található. Az szélturbina modellen való első változtatás esetén létrejön az első Command, majd ezután minden változtatás egy új Step beszúrását eredményezi. A modellen végrehajtom a következő lépéseket:

* Új input elem beszúrása
* A beszúrt elem Sys Id attribútumát 1-re állítása
* Új input elem felvétele 2 –es Sys Id értékkel
* Új alrendszer létrehozása

A lépéseket követően az editor tartalmát elmentem, ennek hatására a Project Explore ablakban megjelenik egy új fájl.

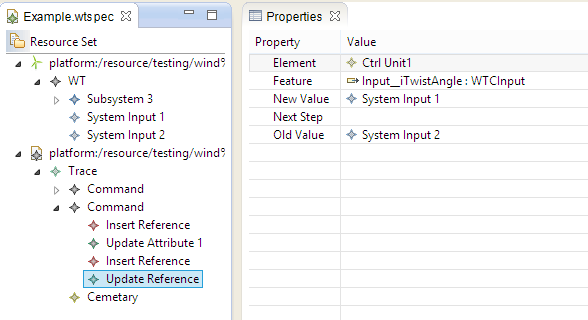


16. ábra: Insert Step a modellen

A 16. ábra látszik, hogy 6 insert műveletet mentett a trace modell. Az Insert Reference típus az elem beszúrását, az Insert Attribute az Id beírását jelöli, mely nem hagyományosan értelmezett beszúró művelet, de az értéke null értékről lett átállítva, ezért én annak kezelem. A kép jobb oldalán látszik a tulajdonságok ablak, melyben az első attribútum beszúrás tulajdonságai látszanak. Az Element mutatja, hogy az 1-es azonosítójú System Input-on történt a módosítás, a Feature mező meg megmutatja, hogy a sysId nevű String típusú attribútuma változott. A new value értékben az „1” értékű string sorosított értéke látszik, míg a következő lépésben, annak referenciája.

Ezt követően a következő parancsokat hajtottam végre:

* Az alrendszerbe 1-es vezérlőegység felvétele
* Az alrendszer Cycle attribútumát átírni 1-be
* Az Input iTwist Angle referenciába berakni az 2-es, majd utána a 1-es bemeneti elemet
* editor mentése

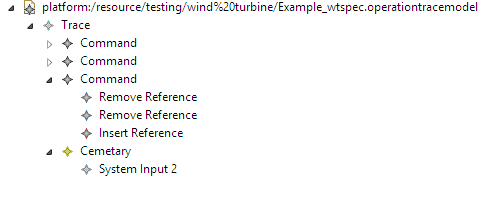


17. ábra: Módosítás a modellen

A 17. ábra mutatja a műveletek eredményét. Az első ami látható, hogy az előző mentés után a Command lezárult, és új Command jött létre. Az első insert Reference a vezérlőegység beszúrását, a második az első értékadást jelzi, amikor az vezérlőegység bemenetét a 2-es bemenetre állítottam. Az Update attribute az Cycle módosításakor, az Update Reference az input váltásakor jött létre. Jobb oldalon ez utóbbi tulajdonságai láthatóak.

Végül végrehajtottam pár törlő műveletet:

* 2-es bemenet kitörlése a turbinából
* Hivatkozás megszüntetése a vezérlőegységben
* Visszavonás művelet használata



18. ábra: Remove művelet

A 18. ábra mutatja, hogy három új lépés jött létre, két remove, és egy insert referencem valamint egy bejegyzés a temetőben. Mind a tartalmazás megszüntetése, mind a referencia kitörlése remove művelet létrejöttét eredményezi, viszont a temetőbe csak a 2-es bemenet került, mivel ennek a tartalmazását szüntettem meg a követett modellben. Az Insert referencia a visszavonás művelet eredménye, ugyanis itt helyezi vissza a rendszer a kitörölt referenciát az 1-es bemenetre.

# Fejezet

Összefoglalás

Szakdolgozatom során megismerkedtem a jelenlegi kollaborációt támogató eszközökkel modellvezérlet szoftverfejlesztő környezetben. Megalkottam egy interfészt, melynek lényege, hogy egy összetett kollaborációs rendszer műveleteit összefogja.

Megterveztem és elkészítettem a Workspace Tracker nevű rendszert, mely ecore modelleken végrehajtott transzformációkat követ nyomon. A nyomon követéshez megalkottam egy szintén Ecore nyelven egy metamoddelt, mely a nyomonkövethetőségi modell struktúráját írja le. Az rendszerrel kapcsolatban a bevezetésben 3 célt tűztem ki:

* Legyen modell független: Az elkészített modell bármely Ecore modell nyomonkövetésére alkalmas azáltal, hogy a modell csupán hivatkozásokat tárol el a modell elemeiről, melyeken a változások végbementek
* Legyen áttekinthető: A nyomonkövethetőségi modell egy egyszerű hierarchiát követő modell, mely csak a legfontosabb adatokat tárolja. A rendszert könnyen bővíthető további műveletekkel.
* Könnyen használható: A felhasználó egyetlen kódsorral be tudja kapcsolni a nyomonkövethetőséget bármely modellen, amennyiben hozzáfér a resourse-hoz. A még könnyebb kezelhetőség érdekében elkészítettem egy Eclipse plugint, melynek segítségével bármely modell editorhoz hozzá lehet adni a nyomononkövetést.

Véleményem szerint az elkészített feladatok teljesítik az előre kitűzött célokat.

# Ábrák jegyzéke

[1. ábra: Ecore felépítése[20] 12](#_Toc406643820)

[2. ábra: ResourceSet felépítése[21] 13](#_Toc406643821)

[3. ábra: Notifikáció 14](#_Toc406643822)

[4. ábra: Offline kollaboráció 17](#_Toc406643823)

[5. ábra: Three-way összefésülés 19](#_Toc406643824)

[6. ábra: Interfész felépítése 21](#_Toc406643825)

[7. ábra: Workspace tracker architektúra 22](#_Toc406643826)

[8. ábra: Modell nyomonkövethetőség 23](#_Toc406643827)

[9. ábra: Nyomonkövethetőségi metamodell 25](#_Toc406643828)

[10. ábra: Beszúrás 26](#_Toc406643829)

[11. ábra: Frissítés 26](#_Toc406643830)

[12. ábra: Eltávolítás 27](#_Toc406643831)

[13. ábra: Áthelyezés 28](#_Toc406643832)

[14. ábra: Szélturbina példánymodell létrehozás 36](#_Toc406643833)

[15. ábra: Adapter hozzáadás 36](#_Toc406643834)

[16. ábra: Insert Step a modellen 37](#_Toc406643835)

[17. ábra: Módosítás a modellen 38](#_Toc406643836)

[18. ábra: Remove művelet 38](#_Toc406643837)

**Irodalomjegyzék**

1. Ábel Hegedűs, István Ráth, Dániel Varró: Replaying Execution Trace Models for Dynamic Modeling. 2013.
2. D1.2 – Evaluation Plan. <http://www.mondo-project.org/>, Projekt dokumentáció (2014.08.22).
3. D4.1 – Collaboration primitives and patterns. MONDO project. <http://www.mondo-project.org/>, Projekt dokumentáció (2014.06.05).
4. Dimitrios S. Kolovos, Louis M. Rose, Nicholas Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan De Lara, István Ráth, Dániel Varró, Massimo Tisi, Jordi Cabot: A Research Roadmap Towards Achieving Scalability in Model Driven Engineering
5. Eclipse Modeling Framework (EMF) - Notifications and Adapters - Tutorial. <http://www.vogella.com/tutorials/EclipseEMFNotification/article.html/>, technológia bemutató (2014.12.13.) .
6. Eclipse. Emfstore project. <http://eclipse.org/emfstore/>, projekt honlap (2014.12.17)
7. Eclipse. <https://www.eclipse.org/>, termék honlap (2014.12.13.)
8. Eclipse. Modeling team framework proposal. <https://www.eclipse.org/proposals/mtf/>, projekt honlap (2014.12.17)
9. Eclipse. The connected data objects model repository (CDO) project. <https://www.eclipse.org/cdo/>, termék honlap (2014.12.17.)
10. ECore documentation <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/package-summary.html#details/>, szoftver dokumentáció (2014.12.13.)
11. Martin Fluegge et al.: Dawn project, <http://wiki.eclipse.org/Dawn/> project honlap (2014.12.17.)
12. Morsa (Model-Oriented Repository for Scalable Access) <http://modelum.es/trac/morsa/>, projekt honlap (2014.12.17)
13. Niels Pinkwart. A Plug-In Architecture for Graph Based Collaborative Modeling Systems. In V. Aleven et al, editor*, Supplementary Proceedings of the 11th Conference on Artificial Intelligence in Education, Sydney (Australia)*, pages 89–94, Sydney, Australia, 2003. SIT.
14. QGears Kft: EMFCollab, <http://qgears.com/products/emfcollab/>,termék honlap (2014.12.17.)
15. The AMOR project. <http://modelversioning.org/>, projekt honlap (2014.12.17.)
16. The Eclipse Foundation. Eclipse Modeling Project. <https://www.eclipse.org/modeling/emf/>, termék honlap (2014.12.13.)
17. The ModelCVS project. <http://modelcvs.org/>, projekt honlap (2014.12.17.)
18. TransactionalEditingDomain documentation <http://download.eclipse.org/modeling/emf/transaction/javadoc/1.1.1/org/eclipse/emf/transaction/TransactionalEditingDomain.html/>, szoftver dokumentáció (2014.12.13.).
19. Xenos, Stefan. Inside the Workbench. A guide to the workbench internals. <https://eclipse.org/articles/Article-UI-Workbench/workbench.html/>, Online cikk (2014. 12 17.)
20. Ecore felépítése <http://download.eclipse.org/modeling/emf/emf/javadoc/2.9.0/org/eclipse/emf/ecore/doc-files/EcoreHierarchy.gif> kép ECore dokumentáció (2014.12.17.)
21. ResourceSet felépítése <http://heod.ru/FILES/02fig09.gif> kép EMF leírás (2014.12.17.)