Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Hozzáférési jogosultságok kezelése kollaboratív modellezésben

Készítette

Papp Krisztián

Konzulens

Dr. Varró Dániel

2014

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

Alulírott Papp Krisztián, 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. 18.

Papp Krisztián

Összefoglaló

Napjainkban összetettebb tervezői folyamatok elengedhetetlen részét képezi a kollaboráció. Azonos modelleken való közös munka elképzelhetetlen megfelelő hozzáférési jogosultság kezelés nélkül. Ennek hatására minden résztvevő csak a szerepköre által meghatározott részegységekkel hajtat végre műveleteket. Ennek nem csak biztonsági szerepei lehetnek, hanem bizonyos szerepkör számára irreleváns rétegek absztrakciója is.

A hagyományos hozzáférési jogosultság kezelés fájlalapú, míg egy összetett modell esetén igény, hogy a jogosultságokat a modellekhez és azoknak tulajdonságaihoz lehessen kötni.

Dolgozatom célja a MONDO projekthez megvalósítani a hozzáférési jogosultságot kezelő modult, illetve a hozzá tartozó kliens oldali offline keretrendszert. A megvalósítás során felhasznált technológiák: EMF (Eclipse Modeling Framework) modellezési keretrendszer és EMF-IncQuery lekérdező nyelv a hozzáférési szabályok lekérdezéséhez. A MONDO projekt célja az összetett modellvezérelt tervező rendszerek számára megkönnyíteni a kollaborációt nagyméretű modellek esetén is.

Abstract

Nowadays, collaboration is necessary part of design process. Organized collaborative design impossible without the right access control. Parts of the model are only visible and for users with the right access rules and also limit the access of the possible actions due to access control. This kind of limitation is not only for security reasons, also for abstraction the irrelevant layers of the model. Because for some kind of user groups a complex model can be transform to a simpler model, due to hide the irrelevant parts from them.

In a usual access control the basic unit is file, but in a complex design based on models required that access control based on models and properties of the model.

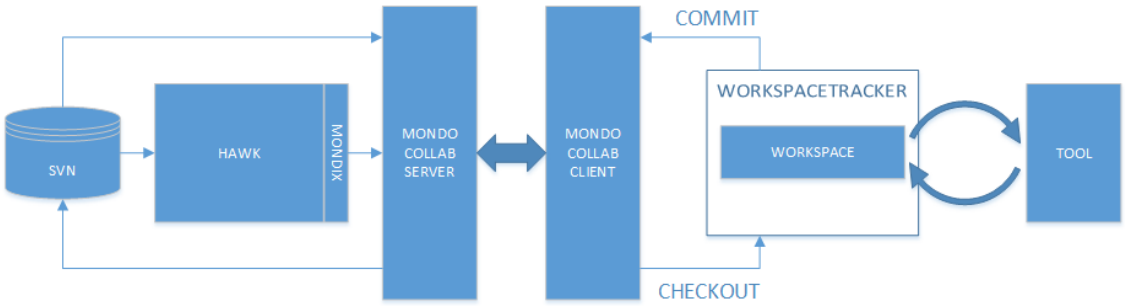
The aim of this document is to provide a user access control module for MONDO project and includes the client side offline framework. Applied technologies: EMF (Eclipse Modeling Framework) and EMF-IncQuery. The aim of the MONDO project to achieve scalability in collaboration in complex model based design frameworks.

1. Bevezető

A dolgozat elkészítésének idejére csatlakoztam a MIT tanszéken futó MONDO [12] projekthez, melyhez tartozó kutatásokat a Hibatűrő Rendszerek Kutatócsoport végez el. A projekt létrejöttét valós ipari partner cégek kezdeményezték. A projekt célja egy összetett modell alapú fejlesztőrendszer kidolgozása, ami kielégíti a partner cégek által felállított követelményeket. A partnerek modellvezérelt fejlesztőrendszereket használnak, melyekben a modellek nagy méretei skálázódási problémákat vetnek fel.

További célja a projektnek, hogy megvalósulhasson egy *Online* és egy *Offline* kollaborációra alkalmas modellező fejlesztőrendszer. Dolgozatomban az Offline megoldásban megjelenő hozzáférési jogosultságkezelést vizsgálom.

A projekthez rajtam kívül több diáktársam is csatlakozott szakdolgozatuk elkészítése céljából. A feladataink elkülönülnek egymástól, viszont előfordulnak bizonyos határterületek, ahol találkoznak ezek a feladatok, amennyiben a dolgozatomban érintek ilyen területet, azt jelzem.



1.1. ábra MONDO projekt Offline kollaboráció egységei

Az **1.1. ábra** által bemutatott rendszer elemeinek leírása:

- TOOL: modellező eszköz, nem feltétlen a projekt keretein belül fejlesztett szoftver.
- WORKSPACE: a modellező eszköz által használt munkaerőforrás.
- WORKSPACETRACKER: a modellező szoftverben a modellen elvégzett módosítások elkapását, észlelését és rögzítését elvégző egység. A *workspacetracker* folyamatosan karbantart egy kimeneti fájlt, ami tartalmazza a végrehajtott modellezési parancsokat.

- CHECKOUT: kollaborációs parancs, lekérdezi és betölti a legfrissebb állapotát a modellnek.
- COMMIT: kollaborációs parancs, feltölti a *workspace*-ben módosított modellt a verziókövető szerverre.
- MONDO COLLAB CLIENT: kollaborációs kliens oldali interfész. Biztosítja, és fogadja a *commit*, *checkout* parancsokat. Amennyiben a kért parancsra jogosult a felhasználó, továbbítja a parancsot a szerver oldali interfésznek.
- MONDO COLLAB SERVER: kollaborációs szerver oldali interfész. Fogadja a kliens oldali interfész parancsait. Biztosítja a verziókövető szerverről a modell lekérdezését, illetve továbbítja a kliens oldal *commit* igényét.
- HAWK+MONDIX: index alapú fájltároló egység szerver oldalon.
- SVN: szerver oldali verziókövető szerver.

1.1. Esettanulmány

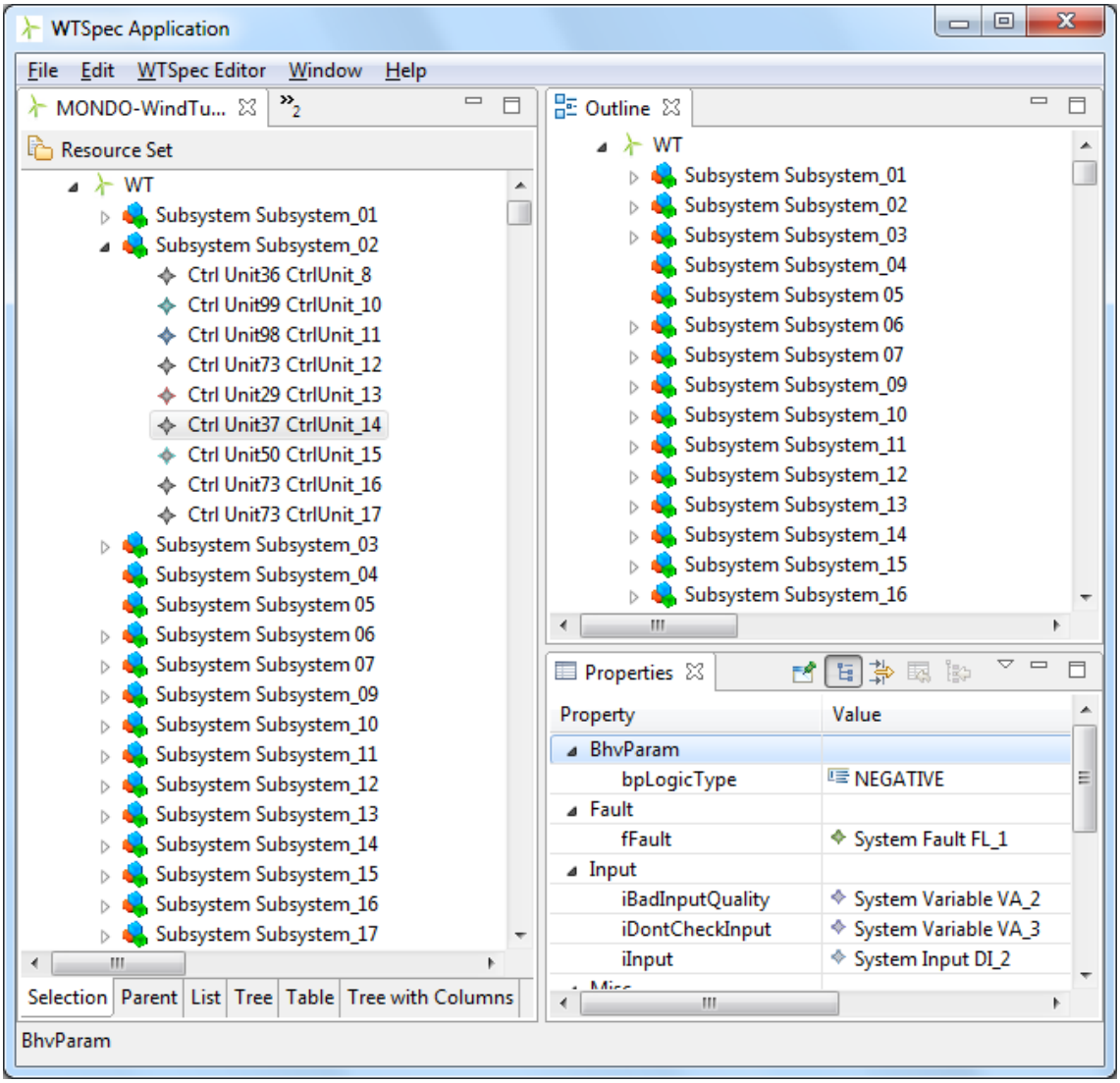
Az Offline kollaboráció igényeit a partner cégek közül az IkerLan cég fejtette ki részletesen, emiatt dolgozatomat ennek a cégnek az esettanulmánya alapján írtam.

Az IkerLan cég szélerőműveket tervez. A tervezéshez egy modell alapú Eclipse fejlesztőrendszert használnak.

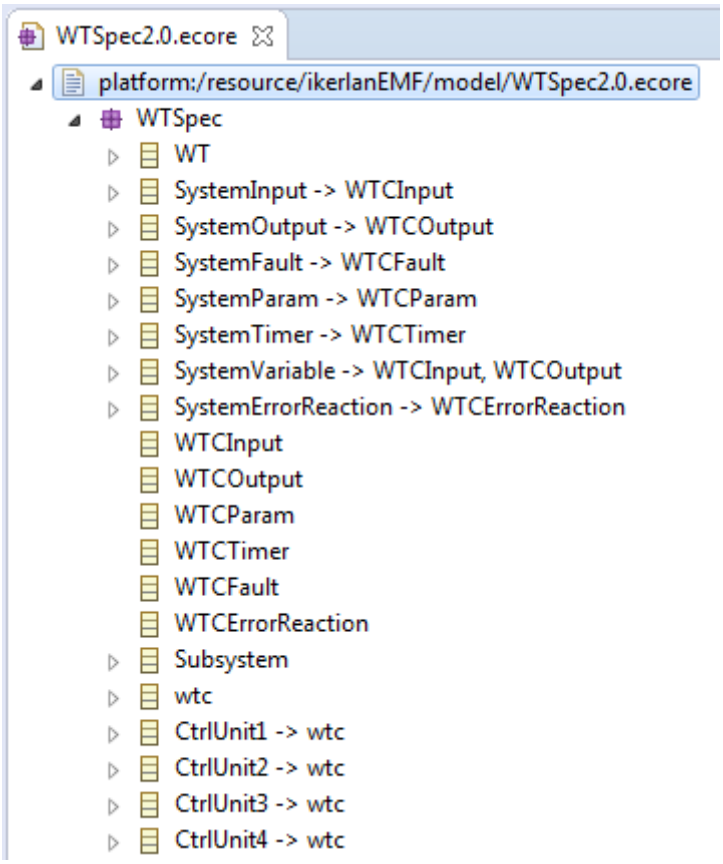
A modellező meta-modelljében az alábbi objektumok találhatóak meg:

- WT (gyökérelem)
- Subsystem (alrendszer elem)
- CtrlUnit (szabályzó elem)
- Rendszerelemek
  - Kimenet
  - Bemenet
  - Paraméterek
  - Riasztások
  - Változók

Minden meta-modell egy WT típusú gyökérelemmel kell kezdődnie. Az alrendszer elemek tetszőleges mélységben tartalmazhatnak további alrendszer elemeket. Rendszer elemeket csak a WT gyökér gyerek elemeiként lehet hozzáadni a modellhez. A szabályzó elemek hivatkozást tartalmaznak rendszer elemekre és csak alrendszer gyerek elemeként hozhatóak létre.



1.2. ábra IkerLan modellező program



1.3. ábra IkerLan modellező eszköz által használt Ecore modell részlete

A modellező eszközben az alábbi műveletek lehetségesek:

- Elem hozzáadása
- Elem eltávolítása
- Elem mozgatása
- Attribútum szerkesztése

Az esettanulmányban két felhasználó különböző jogokkal rendelkezik és ugyan azt a modellt szerkesztik. A két felhasználó „Alice” és „Bob”. Alice nem adhat hozzá vagy törölhet rendszer bemeneti és kimeneti elemet a modellben, viszont Bob elvégezheti ezeket a műveleteket.

Amennyiben Alice elvégzi az alábbi műveleti sort a rendszernek nem szabad engednie a commit-ot.

1. WT.addSystemInput(Input_99)
2. Generator.addCtrlUnit(CtrlUnit_99)
3. CtrlUnit_99.setInput(Input_99)
4. CtrlUnit_99.setOutput(Output_1)
5. Commit (Sikertelen)

Amennyiben csak az alábbi műveleti sort végzi el, sikeresen feltöltheti változtatásait a verziókövető rendszerbe.

1. CtrlUnit_99.setInput(Input_1)
2. Commit (Sikeres)

Bob tetszőlegesen szerkesztheti a rendszer kimeneti és bemeneti elemeit. A műveletsor után sikeresen elvégezheti a commit műveletet.

1. WT.addSystemInput(Input_100)
2. Generator.addCtrlUnit(CtrlUnit_100)
3. CtrlUnit_100.setInput(Input_100)
4. CtrlUnit_100.setOutput(Output_1)
5. Commit (Sikeres)

2. Technológiai áttekintés

Az alábbi fejezetben összefoglalom a dolgozat elkészítése közben felhasznált technológiákat.

2.1. Hozzáférési jogosultságok kezelése

Az információbiztonság egyik alapeleme a hozzáférési jogosultságok kezelése. Ez határozza meg, hogy a felhasználók mely részhalmaza érheti el a szolgáltatásokat vagy erőforrásokat. Bár a jogosultságkezelésnek több technikája is kialakult az informatikában, a dolgozatomban én két technikát fejtek ki részletesebben, ezek a szerep, illetve attribútum alapú jogosultság kezelések.

2.1.1. Szerep alapú hozzáférési jogosultság kezelés (RBAC)

Nagyméretű rendszereknél, hol több felhasználó is azonos jogkörrel rendelkezik a felhasználókat szerepkörökhöz kötik és a hozzáférési jogosultságot nem felhasználónak, hanem szerepköröknek biztosítanak. Ilyen rendszerekben a felhasználónak azonosítania kell magát és aktív szerepköreit. Csak abban az esetben kap hozzáférést az erőforráshoz, amennyiben teljesíti az azonosítást és az azonosított felhasználó aktív szerepkörének engedélyezett a hozzáférés.

Előnyei közé tartozik a széles körű elterjedtség, a megbízhatóság és az egyszerű használat. Viszont a technológia jelentős hátránya a kötöttség. Csak előre definiált és megvalósított szerepkörökkel lehet dolgozni. [1][2]

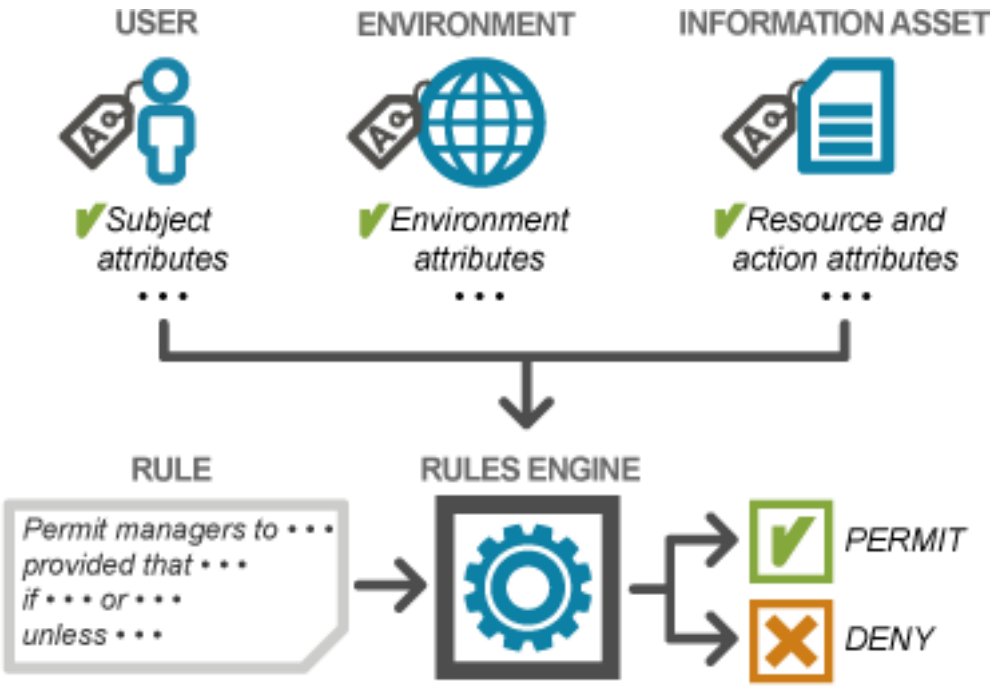
2.1.2. Attribútum alapú hozzáférési jogosultság kezelés (ABAC)

Egy absztrakt technológia, ami tetszőleges attribútumok értékeinek halmaza alapján előállított igényt küld a jogosultság ellenőrzőnek. A jogosultság ellenőrző összeveti az attribútumok értékét a hozzáférés szabályzattal és eldönti az igény elfogadását vagy elutasítását. Az attribútumokat kulcs-érték párokban hordozzák a szükséges információt.

Míg RBAC esetén a felhasználó elküldi a szerepkörét az ellenőrző egységnek, ami csak az alapján dönt. ABAC esetén a felhasználó elküldi a műveletet jellemző összes paramétert attribútumként, amik alapján döntés születik a művelet engedélyezéséről.

Az RBAC egy speciális esete ennek a technológiának, hiszen az egyedüli attribútum ebben az esetben a szerep. Így attribútum alapú hozzáféréssel megoldható egy RBAC rendszer is. [2]

Az attribútum alapú hozzáférés előnyeként említhető meg a rugalmasság. Az attribútumok értékeinek nem csak statikus értékek adhatók, hanem futásidőben generálódott dinamikus adatok is (pl: aktuális idő, földrajzi hely, stb). [1]



2.1. ábra Attribútum alapú hozzáférési jogosultság ellenőrző működését bemutató sematikus ábra [3]

2.2. XACML

Az XACML [4] (eXtensible Access Control Markup Language) egy, az OASIS által létrehozott, leggyakrabban az ABAC rendszerek által használt XML alapú leíró nyelv. 2013-ban megjelent a nyelv 3.0-ás verziója, melyben jelentős formátumbeli változást hoztak a 2.0-ás verzióhoz képest. Dolgozatomban a 3.0-ás verzióval foglalkozok.

Az XACML nyelv által definiálható egységek:

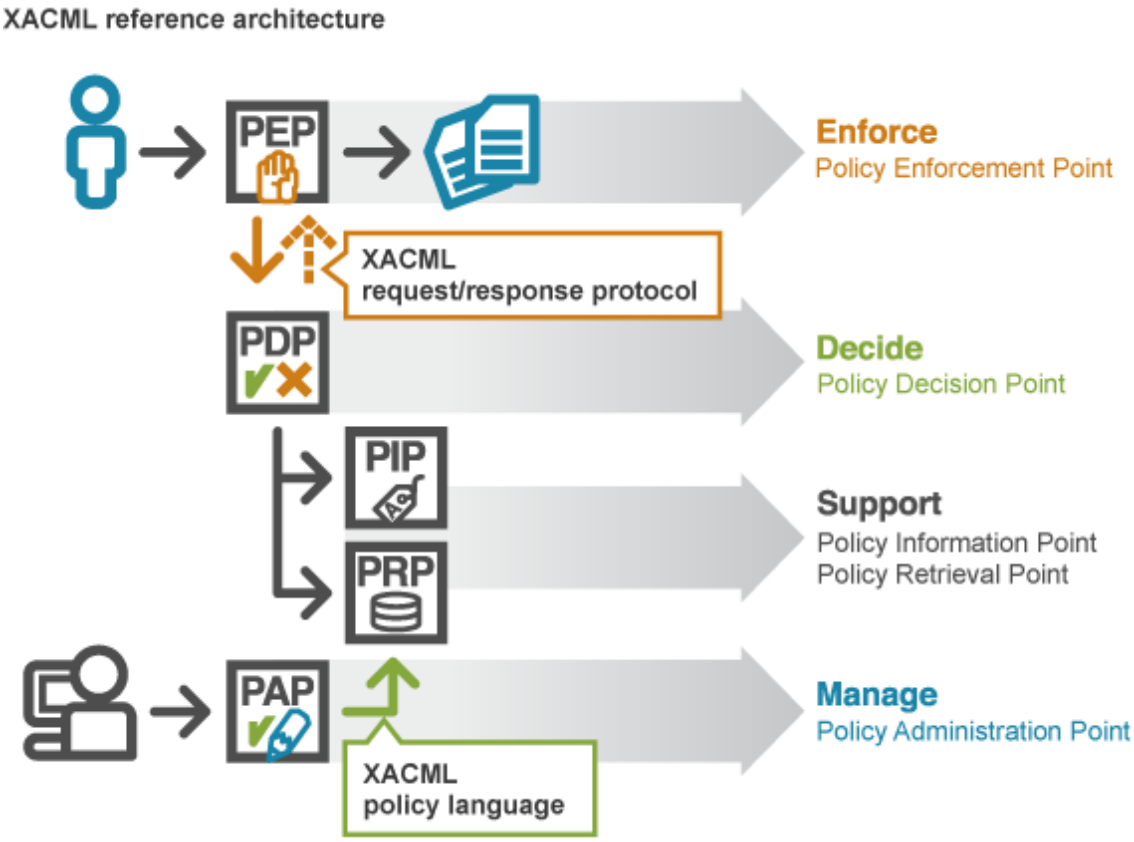
- **Házirend** (Policy): szabályzat, ami meghatározza a hozzáférési jogosultságokat.
- **Kérés** (Request): a felhasználó által küldött kérés tartalmazza műveletet jellemző összes attribútumot.

- **Válasz** (Response): jogosultság ellenőrző által kibocsátott válasz, melyben jelzi, hogy elfogadta, vagy megtagadta a kérést.

Az XACML előnye a kiterjeszthetőség, mivel a nyelv engedi saját típusok és attribútumok definiálását, ezzel megkönnyítve felhasználhatóságát egyedi rendszerekben.

Egy XACML hozzáférési szabályt használó rendszerben az alábbi logikai egységek különülnek el:

- **PEP** (Policy Enforcement Point): Hozzáférési kezelő belépési pontja, melyhez befut a felhasználó által végrehajtandó művelet kérési szándéka. A PEP átalakítja a kérést XACML nyelvű kérésé, amit továbbít a PDP modulnak. Amennyiben a kérést engedélyezett, tovább engedi a felhasználó kérését az erőforráshoz.
- **PDP** (Policy Decision Point): A modul, amelyik a kérést elbírálja és előállítja az XACML nyelvű választ, amit visszaküld a PEP modulnak. A döntéshez összeszedi a szükséges információkat a PIP és PRP moduloktól. A döntést az XACML nyelven írt házirend, a PEP modultól kapott kérés és a PIP modul által biztosított információk által hozza meg.
- **PIP** (Policy Information Point): a PEP által küldött kérésben nem szereplő attribútumhoz biztosít értéket külső forrásból, amit a PDP igényelne a döntéshez.
- **PRP** (Policy Retrieval Point): biztosítja a PEP modul számára, a házirend elérését.
- **PAP** (Policy Administration Point): modul, amin keresztül kezelhető és létrehozható a házirend és hozzáférési szabályok.



2.2. ábra Hozzáférési jogosultság ellenőrző rendszer sematikus ábrája, mely XACML alapú házirendet használ. Az ábrán látható, hogy melyik modul melyik másik modult szolgálja ki információval. [5]

2.3. Axiomatics: ALFA

Az Axiomatics által fejlesztett ALFA egy Eclipse plugin, melynek segítségével egyszerűen készíthetőek XACML nyelvű házirendek. Az ALFA támogatja már az új XACML 3.0 verzióját. Az egyszerűsített szabályíráshoz az Axiomatics kifejlesztett egy egyedi nyelvtant.

2.3.1. Felépítése

Az ALFA plug-in tartalmaz egy Xtext alapú szerkesztőt, melynek előnye, hogy rendelkezik automatikus kiegészítő funkcióval és szerkesztés közben is ellenőrzi a kód érvényességét. A plug-in deklarálja az alapértelmezett kategóriákat, típusokat és attribútumokat. Viszont ha szükséges lehetséges egyedi elemek definiálása is.

Az alábbi példával szemléltethető az ALFA és az XACML közötti különbség. (A példában szereplő szabály az ALFA felhasználói kézikönyv [6] A simple example fejezetében szereplő példa)

Egyedi attribútumok definiálása:

```
namespace Attributes {
    import System.*

    attribute resourceType {
        id = "http://example.com/xacml/attr/resource/type"
        type = string
        category = resourceCat
    }
    attribute resourceClassification {
        id = "http://example.com/xacml/attr/resource/classification"
        type = integer
        category = resourceCat
    }
    attribute userClearance {
        id = "http://example.com/xacml/attr/subject/clearance"
        type = integer
        category = subjectCat
    }
    attribute documentStatus {
        id = "http://example.com/xacml/attr/resource/documentStatus"
        type = string
        category = resourceCat
    }
    attribute documentAuthor {
        id = "http://example.com/xacml/attr/resource/documentAuthor"
        type = string
        category = resourceCat
    }
    attribute subjectId {
        id = "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        type = string
        category = subjectCat
    }
}
```

Házirend definiálása:

```
namespace documents {
    policy topLevel {
        target clause Attributes.resourceType == "document"
        apply denyOverrides
        rule {
            permit
            condition Attributes.userClearance >=
            Attributes.resourceClassification
        }
        rule {
            deny
            condition Attributes.documentStatus == "draft"
            && not(Attributes.documentAuthor == Attributes.subjectId)
        }
    }
}
```

Generált XACML állomány (részlet[1], egy *target* definiálása):

```
<xacml3:Target>
        <xacml3:AnyOf>
            <xacml3:AllOf>
                <xacml3:Match
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <xacml3:AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">document</xacml3:Attribute
Value>
                    <xacml3:AttributeDesignator

AttributeId="http://example.com/xacml/attr/resource/type"
                        DataType="http://www.w3.org/2001/XMLSchema#string"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource"
                        MustBePresent="false"
                    />
                </xacml3:Match>
            </xacml3:AllOf>
        </xacml3:AnyOf>
    </xacml3:Target>
```

A fenti példából látható, hogy egy egyszerű szabály megalkotása is mennyire komplikált struktúrában jelenik meg XACML nyelven, ezért elengedhetetlen egy szerkesztő, ami leegyszerűsíti ezt a feladatot.

Az ALFA nyelv az alábbi elemekből épül fel:

[1] A teljes generált XACML állomány megtalálható a Függelékben [1]

- **policy**: Házirend, ami tetszőleges számú szabályt tartalmaz. A szabályok ütközésénél megadható, hogy a tiltás vagy az engedélyezés az erősebb művelet.
- **policyset**: Házirendek gyűjteménye. A házirend tetszőleges számú házirendet tartalmaz. A házirendek visszatérési értékeik ütközése esetén megadható, hogy a tiltás vagy az engedélyezés az erősebb művelet.
- **rule**: Szabály, valamilyen feltételeknek eleget tevő szabály vagy tiltással, vagy engedélyezéssel tér vissza.
- **target**: A szabály eldöntéséhez szükséges attribútumok halmazát és azoknak konstans értéki megkötéseiket definiálja. Több attribútumokat ÉS, VAGY operátorokkal lehet összekötni. Target-ben az attribútumok értékei függvényekkel is vizsgálhatjuk.
- **condition**: Attribútumok értékeire kényszer állítás. Lehet dinamikus érték vizsgálat például egy másik attribútum értékével lehetséges az összehasonlítás.

2.3.2. Használata

A plug-in sikeres telepítését követően az Eclipse környezetben létre kell hozni egy általános projektet. A projektben a szabályok definiálását „alfa" kiterjesztésű fájlban kell megtenni.

Amint a plug-in észleli az alfa kiterjesztésű fájl megnyitását, hozzáadja a projekthez az Xtext környezetet, ez gondoskodik a nyelvtan ellenőrzéséről. Ahhoz hogy felhasználhassuk a plug-in által definiált alapértelmezett XACML típusokat (pl.: String), ahhoz hozzá kell adni a projekthez a telepítőben megtalálható *standard-attributes.alfa* és *system.alfa* fájlokat. Ezek a fájlok tartalmazzák az alapértelmezett típus, attribútum, kategória és függvény és operátor definíciókat. A nyelv lehetőséget biztosít az alapértelmezett elemeket kiegészítve sajátokat is definiálni.

2.4. wso2: Balana

A korábbi fejezetben bemutatásra került az XACML alapú jogosultság ellenőrzés. Két fő eleme van, a házirend és a hozzáférési kérelem, mindkettő statikus XACML tartalom. A PDP modul döntéshozatalkor, ezeket hasonlítja össze. A WSO2 által fejlesztett Balana [7] projekt egy nyílt forráskódú Java alapú implementációja az XACML nyelvnek, viszont magában foglalja egy PDP modul megvalósítását is. A forrásállomány elérhető GitHub-ról.

Több Java nyelvű XACML implementáció létezik, viszont jelentős hiányosságuk az elavultság. A projektek nagy része még nem támogatja az XACML 3.0-ás változatát, viszont a Balana már az XACML 3.0-át támogatja.

Előnyei közé tartozik, az egyszerű használat, viszont jelentős hátránynak találtam a bővíthetőség hiányát. Míg az ALFA kiegészítése egyéni kategóriákkal, függvényekkel egyszerűen megvalósítható, addig a Balana rendszerében komplikáltabb.

2.5. Eclipse

Az Eclipse Foundation [8] által fejlesztett nyílt forráskódú, platform független keretrendszer. Legelterjedtebb felhasználási módja a Java IDE fejlesztőrendszer. A keretrendszer sikerei miatt más programozási nyelvek integrált fejlesztőkörnyezeteként is használatos. Az alap program kiegészítése Eclipse plug-inek segítségével lehetséges.

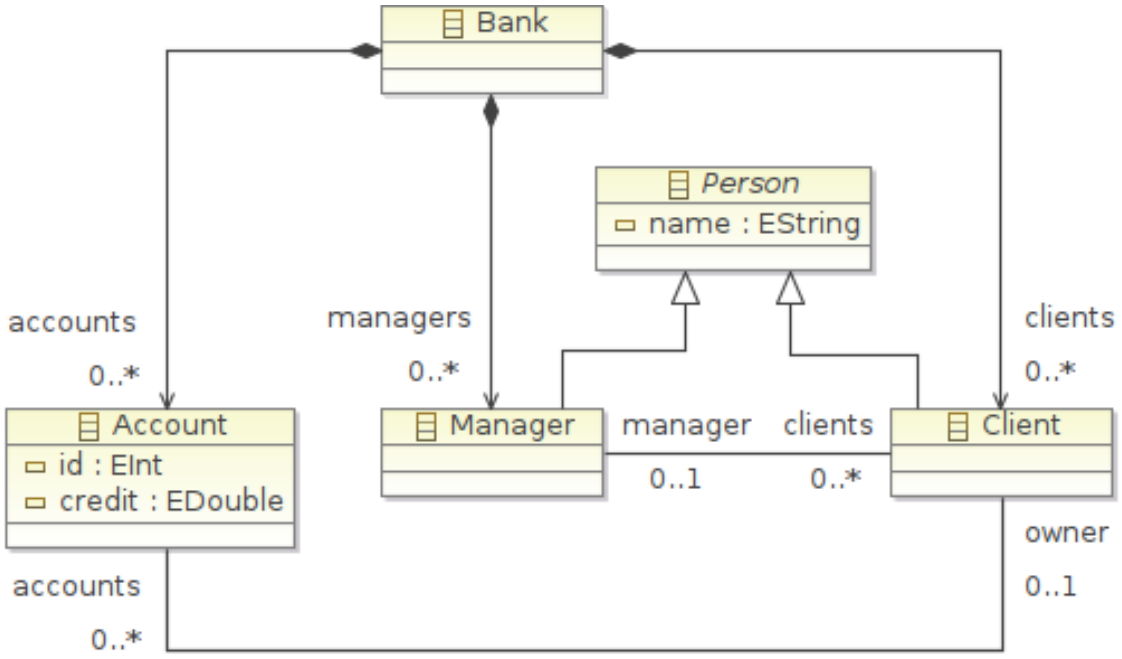
2.6. Eclipse plug-in fejlesztés

Két típusú plug-in létezik az RCP (Rich Client Platform) és az RAP (Rich Ajax Platform). RCP segítségével egy független futtatható alkalmazást állítható elő. Az Eclipse egy plug-in alapú keretrendszer, vagyis azoknak a funkcióknak a nagy része, amit plug-in fejlesztésre használunk szintén egy plug-in ként jelennek meg a keretrendszerben. Az plugin-ek csak akkor töltődnek be a keretrendszerbe, ha szükséges (*Lazy loading*), ezt figyelemben kell tartani fejlesztés közben. A plug-in meta információi a *MANIFEST.MF*, illetve a *plugin.xml* fájlokban tárolódnak. Ilyen meta információ például a plug-in azonosító, más plug-inektől való függőség. A plug-inek egymásra épülhetnek, amik előre definiált *extension point*-on keresztül valósulnak meg.

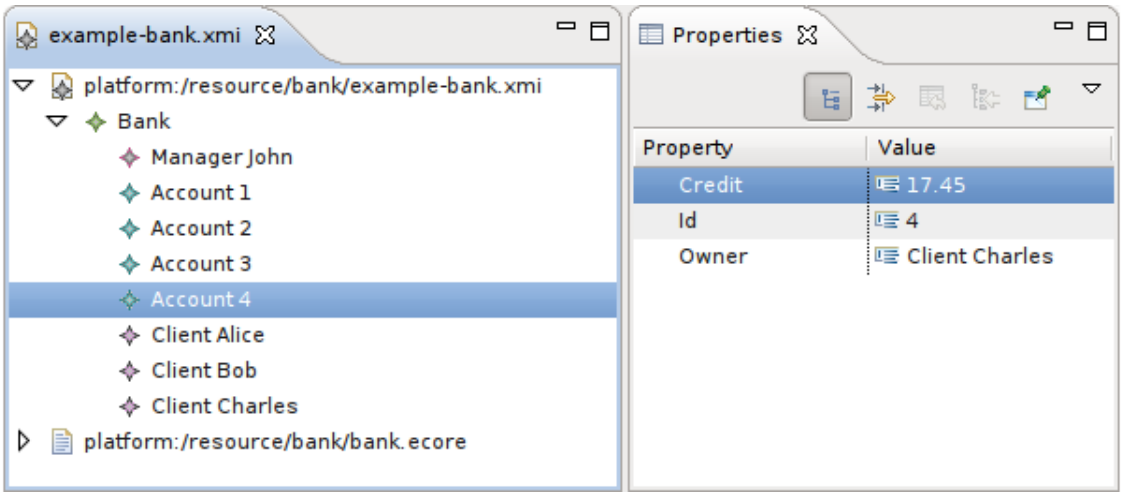
2.7. Eclipse Modeling Framework (EMF)

Az EMF keretrendszer lehetőséget biztosít modellező fejlesztőrendszerek készítésére. A modellek a *meta-modellen* alapulnak. A meta-modell az UML osztálydiagramjához hasonlóan egy olyan jellegű leírás, ami meghatározza, hogy milyen típusú modell elemek hol tárolódhatnak, illetve melyik entitás mely más entitásokkal áll kapcsolatban. A meta-modell elkészítését követően a keretrendszer elvégzi a meta-modellt leíró osztályok generálását. [9]

A meta-modelleket az *Ecore* meta-modell leíró nyelv segítségével definiálhatjuk. Ecore modell több féle képen létrehozható, grafikus szerkesztővel vagy úgynevezett *Tree Edior*ral.



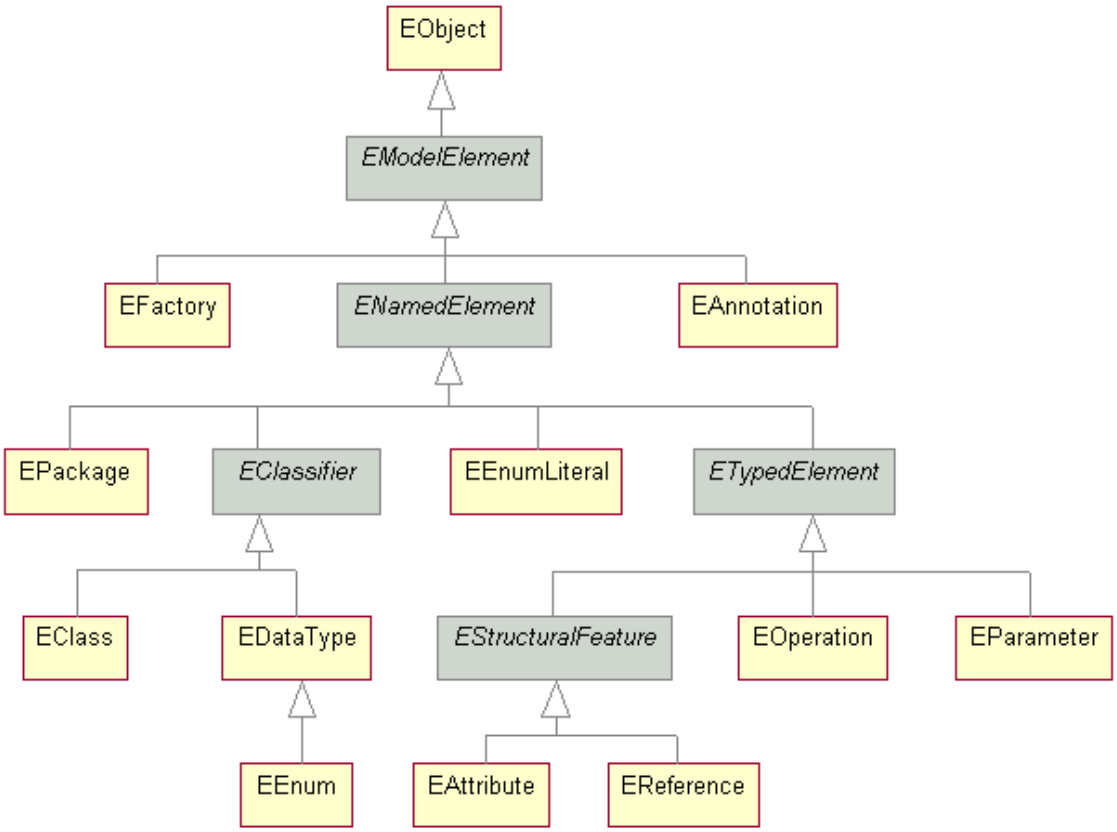
2.3. ábra Ecore diagram grafikus szerkesztőfelülete és a diagram vizualizációja [11]



2.4. ábra Ecore modell Tree Editor nézete meta-modell szerkesztés közben.

Az EMF a modellezés közben folyamatosan generálja a meta-modellt implementáló osztályokat. A modellezés befejeztével az ecore fájlból generálható az Edit és Editor projektek. Ezeknek a projektek segítségével elkészíthető a meta-modellen alapuló modellező fejlesztőrendszer plug-in megvalósítása.

Ecore modell készítéséhez az alábbi osztályok állnak rendelkezésre:



2.5. ábra Eclipse EMF Ecore modellt felépítő osztályok hierarchiája[10]

3. Hozzáférési jogosultság ellenőrző plug-in

3.1. Tervezői döntések

A projekt igényei alapján a hozzáférési jogosultságokat a modell attribútumai alapján is meg lehet adni, ezért az ABAC hozzáférési technológiát terveztem megvalósítani. Viszont érvényesíteni akartam az RBAC előnyét is a jogok szerepek körökhöz való kötésével, mivel így egyszerűbben lehet jogot adni a felhasználóknak. Emiatt egy attribútum alapú jogosultság kezelő alapjait terveztem meg, ami támogatja a szerep alapú jogok meghatározását.

3.2. Hozzáférési szabályzat

3.2.1. Bevezetés

Mivel attribútum alapú hozzáférési jogosultság ellenőrző technológiát valósít meg a kollaborációs modul, ezért a szabályzatot XACLM nyelven fogalmazom meg. Az XACML szabályzat definiálási folyamatának egyszerűsítése céljából használtam fel az Axiomatics által Eclipse keretrendszerhez fejlesztett ALFA plug-int.

A szabályok létrehozásának előfeltétele a jogosultság eldöntéséhez szükséges attribútumok definiálása. A dolgozat keretén belül felépítettem azt az attribútum halmazt, mellyel a későbbi felhasználás esetén új szabályok definiálhatók.

XACML nyelvtanban egy attribútum három tulajdonsággal rendelkezik:

- Kategória azonosító: előre definiált kategória azonosító. XACML által támogatott beépített kategóriák:
    - **Subject category**: a jogosultság kérés alanya, vagyis a felhasználó
    - **Resource category**: az erőforrás, amihez az alany hozzá kíván férni
    - **Action category**: a művelet, amit az alany végre kíván hajtani
    - **Enviroment category**: egyéb környezeti paraméterek
- Attribútum azonosító: hivatkozási lehetőség a konkrét attribútum elemre.
- Típus: előre definiált típusa az attribútum értékének, például szöveg tartalom esetén string.

Az XACML nyelvben ezeknek az értékeknek definiáláshoz a kategóriák és típusok azonosítóit kell használni, ami jelentősen megnehezíti a szabály írásának és olvashatóságának. Az ALFA nyelvtan elrejti ezeket az azonosítókat. A szabályíráshoz szükséges attribútumok definiálása ALFA nyelven az alábbi képen néz ki.

```
namespace user{
    attribute role{
        category = subjectCat
        id = "role"
        type = string
    }
    attribute userId{
        category = subjectCat
        id = "urn:oasis:names:tc:xacml:1.0:subject:subject-id"
        type = string
    }
}

namespace action{
    attribute actionId{
        category = actionCat
        id = "urn:oasis:names:tc:xacml:1.0:action:action-id"
        type = string
    }
}

namespace resource{
    attribute resourceType{
        category = resourceCat
        id = "resource-type"
        type = string
    }
    attribute resourceAttributeName{
        category = resourceCat
        id = "resource-attribute-name"
        type = string
    }
    attribute resourceAttributeValue{
        category = resourceCat
        id = "resource-attribute-value"
        type = string
    }
}
```

A logikailag összetartozó attribútumokat közös névtérben definiálhatóak. Az *attribute* parancs után deklaráljuk az attribútum nevét, amin keresztül hivatkozunk rá az ALFA nyelven írt szabályokban. Kategória meghatározásánál felhasználhatóak az XACML által beépített kategóriák, amiket az ALFA alapértelmezetten kezel. Az attribútum azonosító esetén lehetőség van egyedi, új attribútum definiálásra, vagy lehet hivatkozni egy alapértelmezett attribútumra (lsd: a userId egy alias a beépített *subject-id*-ra).

A felhasználó azonosítására kétféle lehetőséget biztosítottam, vagy szerepkör, vagy felhasználói azonosító alapján. Így lehetőség van felüldefiniálni egyes felhasználók jogosultságát. Művelet esetén egyedül csak a művelet típusára lehet szűrni. Az erőforrás a szerkesztő rendszer modelljének egy eleme lehet. Ahhoz, hogy beazonosítható lehessen egy modell elem, ahhoz a típusát és/vagy valamely attribútumainak értékét kell meghatározni. Nem tüntettem ki az elem azonosítóját külön attribútumként, mivel mindig a modell válogatja meg, hogy mikor is értelmes az azonosító kifejezés. Az attribútumokra kulcs-érték párokban lehet hivatkozni.

3.2.2. Szabályzat írás

Az IkerLan esettanulmány alapján olyan szabályzatot definiáltam, amiben két különböző szerepkör található, különböző jogosultságokkal. ALFA nyelven történő szabályzat definiálást az alábbiak alapján lehet végrehajtani.

Egy atomi szabályt a *rule* kifejezés definiál. Egy szabálynak kétféle visszatérési értéke lehet, az engedélyezés (*permit*) és az elutasítás (*deny*). Ezen felülül opcionálisan hozzáadható feltételvizsgálat, ami például az XACML kérésben szereplő attribútumokra vonatkozhat. A feltételvizsgálatnak két módja található meg az XACML-ben, az egyik a *target clause*, a másik a *condition* kifejezésekkel lehetséges. Jelentős különbség, hogy a *target*-ként megadott feltételek csak konstans kifejezésekkel dolgozhatnak, míg *condition* értékének más attribútum dinamikus értékét is feltételnek adhatjuk.[2]

Szabályok sorozata házirendbe gyűjthetők össze, erre a *policy* paranccsal van lehetőség. Ha a házirend minden szabályának azonos a feltételvizsgálata, akkor a *target clause* vizsgálatot ki lehet emelni a szabályokból. Mivel a házirendben több szabály is szerepelhet, ezért fennáll a lehetősége, hogy a szabályok egymásra ellentétes értékekkel térnek vissza. A házirendeknek meg kell adni, hogy ebben az esetben, hogy oldják fel az ütközést, ez egy egyesítő algoritmus kiválasztásával érthető el. Az XACML alapértelmezetten rendelkezik néhány egyesítő algoritmussal, ilyen például a *permitOverrides, denyOverrides* vagy a *permitUnlessDeny*. Az *apply* paranccsal van lehetőség egyesítő algoritmus meghatározására.

[2] Axiomatics. *ALFA Plugin for Eclipse User's Guide* (2013) 21. oldal

A házirendek házirend gyűjteményekbe szedhetőek össze, ami a *policyset* paranccsal érhető el. Házirend gyűjtemények tetszőleges számban tartalmazhatnak házirendeket, illetve más házirend gyűjteményeket, amivel hierarchikus szabályok előállítására van lehetőség.

Az alább feltűntetett szabály elsődlegesen lehetőséget biztosít felhasználókhoz azonosító alapján történő egyedi jogosultság hozzárendelését. A kódban az „admin" azonosítóval rendelkező felhasználó számára nincs tiltás a modellben. Két szerepkörhöz tartozó szabály található még meg, az egyik a „*designer*", a másik a „*editor*" azonosítóval ellátott felhasználó szerepkörökhöz kötődik. Az editor jogkörrel rendelkező felhasználó nem szerkesztheti, nem törölheti, vagy nem hozhat létre új példányát a „SystemInput" és a „SystemOutput" elemeknek.[3] Míg a designer jogkörrel rendelkezők szabod szerkeszthetnek modellen, kivéve egy *SystemOutput* típusú elemet, aminek *sysId*-ja „lockedElement" értékű, ezt az elemet nem törölhetik.

A szerkesztő és a létrehozott szabályzat megtalálható a melléklet CollaborationPolicyAdministrationPoint projektben. Az alábbi kódrészlet mutatja fő házirendet. A kódban látható, hogy a nyelvtan lehetőséget ad hierarchikus szabályok létrehozására. Továbbá visszatérési attribútum értékadásával magyarázat adható, a kérelem küldőjének az esetleges elutasítás vagy engedélyezés okáról, körülményeiről.

[3] Az IkerLan esettanulmányban szereplő Alice felhasználó a rendszeremben editor jogkörrel rendelkezik.

```
policyset topLevel {
    apply permitOverrides

    policy userBasedPolicy {
        target clause user.userId == "admin"
        apply permitOverrides
        rule {
            permit
            on permit {
                advice displayAdvice {
                    message = "Permission granted by user based policy"
                }
            }
        }
    }

    policyset roleBasedPolicy{
        apply permitOverrides

        policy designerPolicy {
            target clause user.role == "designer"
            apply denyOverrides
            rule {
                target clause resource.resourceType == "SystemOutput" and
resource.resourceAttributeName == "sysId" and resource.resourceAttributeValue
== "lockedElement" and action.actionId == "RemoveReference"
                deny
            }
            rule {
                permit
            }
            on permit {
                advice displayAdvice {
                    message = "Permission granted by role based policy:
designer policy"
                }
            }
            on deny {
                advice displayAdvice {
                    message = "Permission denied by role based policy:
designer policy"
                }
            }
        }

        policyset editorPolicy {
            target clause user.role == "editor"
            apply denyOverrides
            policy {
                target clause resource.resourceType == "SystemInput" or
resource.resourceType == "SystemOutput"
                apply denyOverrides
                rule {
                    condition action.actionId == "InsertReference" ||
action.actionId == "RemoveReference"
                        || action.actionId == "MoveReference"
                     deny
                     on deny {
                        advice displayAdvice {
```

```
                            message = "Designers doesn't have access to
SystemOutputs or SystemInputs!"
                        }
                    }
                }
                rule {
                    permit
                }
            }

            on permit {
                advice displayAdvice {
                    message = "Permission granted by role based policy: edi-
tor policy"
                }
            }
            on deny {
                advice displayAdvice {
                    message = "Permission denied by role based policy: editor
policy"
                }
            }
        }
    }
}
```

3.3. Felhasználó szerkesztői lépéseinek nyomon követése[4]

Ahhoz, hogy el lehessen dönteni egy módosított modellről, hogy megsért-e bármilyen jogosultságköri szabályt, ismerni kell pontosan a felhasználó által elvégzett szerkesztés minden egyese lépését. Amennyiben ismertek a szerkesztési lépések, lefuttatható rajtuk a jogosultság ellenőrző lekérdezések. A felhasználó szerkesztői lépések nyomon követését végzi el az úgynevezett *WorkspaceTracker [13]*. A WokspaceTracker a projekten dolgozó egyik hallgatótársam munkája, jelen alfejezetben összefoglalom a modul működését, mivel a hozzáférési jogosultság ellenőrző plug-in elengedhetetlen részét képezi ez a modul.

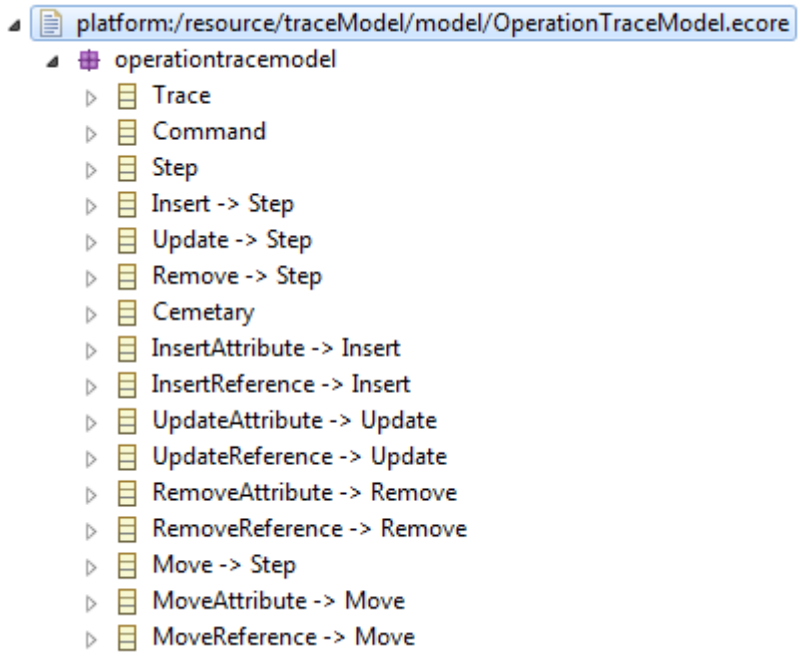
A WorkspaceTracker feliratkozást kezdeményez az Eclipse keretrendszer forrásfájlt nyomon követő eseményekre. Ez alapján a modul mindig értesül amennyiben változás jelent meg a beregisztrált forrásfájlban. A modul detektálja az egyes lépéseket, megállapítja azokról, hogy milyen típusú módosítást jelent a forrásmodellen. A sikeres beazonosítás után eltárolódik egy *operationaltracemodel* (3.1. ábra) példányban.

A feldolgozás közben felépít egy adatstruktúrát, egy listát, mely *Command*-okból áll. A *Command* tartalmazza két forrásfájl mentése közti módosítások sorozatát. Tehát a

[4] Ebben az alfejezetben nem saját fejlesztésű modult mutatok be.

Command tartalmaz egy kollekciót, amiben megtalálhatóak a módosításokat, amik a *Step* osztály leszármazottjai. A *Step* egy absztrakt ősosztály a konkrét leszármazottak szimbolizálják a különböző típusú módosításokat. Az adatstruktúra olyan további segéd attribútumokkal és függvényekkel van felépítve, amik segítségével pontosan bejárható sorrendhelyesen az egyes módosítási lépések.

Az összes módosítás feldolgozása után a szerializálható adatstruktúra elmentésre kerül az Eclipse szerkesztő program gyökér könyvtárában.



3.1. ábra Az operationtracemodel ecore modellje

```
<?xml version="1.0" encoding="ASCII"?>
<operationtracemodel:Trace xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:ecore="http://www.eclipse.org/emf/2002/Ecore"
xmlns:operationtracemodel="http://operationtracemodel/1.0"
xmlns:wtc="http://WTSpec/2.0" firstCommand="//@Commands.1"
lastCommand="//@Commands.2">
  <Commands nextCommand="//@Commands.2" firstStep="//@Commands.1/@Steps.0">
    <Steps xsi:type="operationtracemodel:InsertReference"
nextStep="//@Commands.1/@Steps.1">
      <element href="platform:/resource/teszt2/file.wtspec#/"/>
      <feature xsi:type="ecore:EReference"
href="http://WTSpec/2.0#//WT/itsSubsystems"/>
      <newValue href="plat-
form:/resource/teszt2/file.wtspec#//@itsSubsystems.6"/>
    </Steps>
    <Steps xsi:type="operationtracemodel:InsertReference"
nextStep="//@Commands.1/@Steps.2" newValue="//@cemetary/@removedElements.1">
```

```
      <element href="plat-
form:/resource/teszt2/file.wtspec#//@itsSubsystems.6"/>
      <feature xsi:type="ecore:EReference"
href="http://WTSpec/2.0#//Subsystem/itsWTCs"/>
    </Steps>
    <Steps xsi:type="operationtracemodel:InsertReference"
nextStep="//@Commands.2/@Steps.0" element="//@cemetary/@removedElements.1">
      <feature xsi:type="ecore:EReference"
href="http://WTSpec/2.0#//CtrlUnit23/Input__iInverterPower"/>
      <newValue href="platform:/resource/teszt2/file.wtspec#//@itsInputs.0"/>
    </Steps>
  </Commands>
  <Commands firstStep="//@Commands.2/@Steps.0">
    <Steps xsi:type="operationtracemodel:RemoveReference"
oldValue="//@cemetary/@removedElements.1">
      <element href="plat-
form:/resource/teszt2/file.wtspec#//@itsSubsystems.6"/>
      <feature xsi:type="ecore:EReference"
href="http://WTSpec/2.0#//Subsystem/itsWTCs"/>
    </Steps>
  </Commands>
  <cemetary>
    <removedElements xsi:type="wtc:SystemParam"/>
    <removedElements xsi:type="wtc:CtrlUnit23">
      <Input__iInverterPower xsi:type="wtc:SystemInput" href="plat-
form:/resource/teszt2/file.wtspec#//@itsInputs.0"/>
    </removedElements>
  </cemetary>
</operationtracemodel:Trace>
```

A fenti kódrészlet bemutatja a WorkspaceTracker által nyomon követett modell módosulásait tartalmazó fájl tartalmát.

3.4. Felhasználó szerkesztői lépéseinek feldolgozása

Amennyiben rendelkezésre áll az előző fejezetben bemutatott követő fájl, az azt jelenti, hogy a modellen változás következett be. A változások feldolgozásához a modulomnak fel kell dolgozni az elmentett *operational trace modelt*. A feldolgozás során a program azonosítja a felhasználó módosítását, megkeresi az érintett elemet. Amennyiben nem attribútum került szerkesztésre, abban az esetben az elem összes attribútumát is lekérdezi, mivel ezek alapján az attribútumok alapján lehetséges csak egy elem pontoson meghatározása (például *sysId* attribútum alapján történő hivatkozással). Ezekkel a tulajdonságokkal eltárolásra kerül egy kollekcióban az összes szerkesztői lépés. A szerkesztői lépések tárolására egy saját osztály példányait használom, a *StepByUser* osztályét. A *StepByUser* osztály példányaiban már csak azokat az adatok kerülnek tárolásra, ami a jogosultság ellenőrzésénél szükséges:

- A módosítást végző felhasználó azonosítója
- A felhasználói művelet azonosítója
- Az érintett elem típusa
- Az érintett elem attribútumainak listája

Az alábbi program részlet illusztrálja a feldolgozás műveletét. A kódrészletben az új elem beszúrása és az új attribútum beszúrása művelet kerül feldolgozásra. Mivel a WorkspaceTracker által előállított fájlt dolgozok fel, ezért a WorkspceTracker modellje által ismert összes művelet feldolgozására fel kellett készítenem a feldolgozó modult, így az alábbi műveletek kerülhetnek feldolgozásra:

- Új elem hozzáadása
- Új attribútum hozzáadása
- Meglévő elem módosítása
- Meglévő attribútum módosítása
- Elem törlése
- Attribútum törlése
- Elem mozgatása
- Attribútum mozgatása

```
while (command != null) {
    Step step = command.getFirstStep();
    while (step != null) {
        String stepType = step.eClass().getName();

        switch (stepType) {
        case "InsertReference": {

            InsertReferenceImpl insert = (InsertReferenceImpl) step;

            EObject newValue = insert.getNewValue();
            EClass newValueClass = newValue.eClass();
            String newValueClassName = newValueClass.getName();

            ArrayList<StepAttribute> attributes = new
ArrayList<StepAttribute>();
            EList<EAttribute> eAllAttributes = newValue.eClass()
                    .getEAllAttributes();
            for (EAttribute eAttribute : eAllAttributes) {
                Object resultingDataType = newValue
                        .eGet(eAttribute);
                if (resultingDataType != null) {
                    attributes.add(new StepAttribute(eAttribute
                            .getName(), resultingDataType
                            .toString()));
                }
            }

            stepQueue.add(new StepByUser(user, stepType,
                    newValueClassName, attributes));

            break;
        }
        case "InsertAttribute": {

            InsertAttributeImpl insert = (InsertAttributeImpl) step;

            EClass elementClass = step.getElement().eClass();
            String elementClassName = elementClass.getName();
            Object newValue = insert.getNewValue();
            String attributeName = step.getFeature().getName();

            EList<EAttribute> eAllAttributes = step.getElement()
                    .eClass().getEAllAttributes();
            ArrayList<StepAttribute> attributes = new
ArrayList<StepAttribute>();

            for (EAttribute eAttribute : eAllAttributes) {
                Object resultingDataType = step.getElement().eGet(
                        eAttribute);
                if (resultingDataType != null) {
                    attributes.add(new StepAttribute(eAttribute
                            .getName(), resultingDataType
                            .toString()));
                }
            }

            stepQueue.add(new StepByUser(user, stepType,
                    elementClassName, attributes));
```

```
                break;
            }

            }

            step = step.getNextStep();
        }
        command = command.getNextCommand();
}
```

3.5. Jogosultság ellenőrző kérelem előállítása és kiküldése

A szerkesztői lépések összegyűjtése után előállíthatóak a jogosultság ellenőrző kérések. A kérés sablonját XACML nyelven írtam meg és a kérés összeállításakor csupán behelyettesítésre kerülnek az aktuális attribútum értékek.

Az *XACMLHelper* osztály *createXACMLRequest* statikus metódusa felelős a végeleges XACML kérelem összeállításáért. Alább látható egy kérelem, melyben „admin" azonosítójú felhasználó törölt egy „CtrlUnit23" típusú elemet.

A kérelem kiküldéséhez szükséges metódusok a *PEPHelper* osztályban kerültek implementálásra. A kiküldés során a kész kérelmet továbbítja egy PDP példánynak.

```
<Request xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
CombinedDecision="false" ReturnPolicyIdList="false">
<Attributes Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-
subject">
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
IncludeInResult="false">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">admin</AttributeValue>
</Attribute>
</Attributes>
<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:action">
<Attribute AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"
IncludeInResult="false">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">RemoveReference</Attribute
Value>
</Attribute>
</Attributes>
<Attributes Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource">
<Attribute AttributeId="resource-type" IncludeInResult="false">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">CtrlUnit23</AttributeValue
>
</Attribute>
<Attribute AttributeId="resource-attribute-name" IncludeInResult="false">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">cycle</AttributeValue>
</Attribute>
<Attribute AttributeId="resource-attribute-name" IncludeInResult="false">
```

```
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">priority</AttributeValue>
</Attribute>
<Attribute AttributeId="resource-attribute-value" IncludeInResult="false">
<AttributeValue
DataType="http://www.w3.org/2001/XMLSchema#string">0</AttributeValue>
</Attribute>
</Attributes>
</Request>
```

3.6. Kérelem feldolgozása

A kérelem feldolgozását a PDP modul végzi. A plug-inben lévő Balana alapú PDP modul használatához szükséges metódusok a *PDPHelper* osztályban találhatóak. A plug-in indulásakor inicializálódik a statikus PDP példány. A modul helyes működéséhez hozzáférést kell biztosítani a modulnak az aktuálisan érvényben lévő XACML házirendhez. Ehhez mindössze meg kell adni a házirend XML fájlokat tartalmazó mappa elérési útvonalát. A plug-in az Eclipse példány gyökérkönyvtárában keresi a *resources* mappát, amiben tetszőleges névvel szerepelhetnek a házirend fájlok.

A kérelem elküldése hatására a PDP példány lefuttatja a kérelem és a házirend egybevetését. Viszont a kiértékelés előtt a PDP beszerzi a számára ismeretlen attribútum értékeket, jelen esetben a felhasználó szerepköreit. A feladatot elméletileg a PIP modul végzi az XACML absztrakt rendszerében, viszont jelen esetben ez a modul nem jelenik meg külön elemként. A Balana típusú PDP modul inicializálásakor lehetőség van úgynevezett *AttributeFinderModule* regisztrálására. Ez a keresőmodul, akkor kerül meghívásra, mikor a házirend feldolgozás közben, olyan attribútumot talál, amit nem definiál a kérelem. A PDP modul is ekkor venné igénybe a PIP modul adatgyűjtő szolgáltatását, tehát *AttributeFinderModule* regisztrálással hozzáadható a rendszerhez egyedi PIP modul működésével ekvivalens működésű modul. A kereső modul megkeresi az kérelem által meghatározott attribútumok közül azon attribútumok halmazát, melyek alapján egyértelműen pótolható a hiányzó paraméter. A programomban a felhasználói szerepkör értéke hiányzik, viszont a szerep alapú házirendben hivatkozva van ilyen attribútumra. Az attribútum kereső modult beregisztráltam a szerepkör azonosítójára. Futása alatt meg kell keresni az attribútumok közül a felhasználói azonosítót, aminek alapján lekérdezhető a felhasználói adathalmazból a felhasználó szerepkörei. Ezeket az értékeket hozzá kell adni a találati attribútum kollekcióhoz (XACML terminológiájában *Bag*).

Az attribútum kereső *findAttribute* metódusa akkor kerül meghívásra, mikor a kérelem feldolgozó megtalálta a felkonfiguráláskor beállított azonosítóval rendelkező attribútumot. Ekkor lefutnak az egyedi attribútum értékeket begyűjtő programrészletek és az eredményül kapott új attribútum értékeket hozzá kell adni a PDP által tárolt attribútumok közé.

Az alábbi kódrészlet tartalmazza az extra attribútum érték keresésének menetét.

```
EvaluationResult result = context.getAttribute(searchType, searchId, issuer,
searchCategory);
if(result != null && result.getAttributeValue() != null &&
result.getAttributeValue().isBag()){
    BagAttribute bagAttribute = (BagAttribute) result.getAttributeValue();
    if(bagAttribute.size() > 0){
        String userName = ((AttributeValue)
bagAttribute.iterator().next()).encode();

        User user = UserDataBase.getUser(userName);
        List<Role> roles = user.getRole();
        for(Role role : roles){
            roleNames.add(new StringAttribute(role.getRoleId()));
        }
    }
}

if (roleNames != null && roleNames.size() > 0) {
    attributeValues.addAll(roleNames);
}

return new EvaluationResult(new BagAttribute(attributeType,
attributeValues));
```

3.7. Felhasználói felület

A plug-in felhasználó felületére ki kell vezetni a kollaborációt irányító vezérlő elemeket. A kollaborációt jelen esetben két paranccsal lehet vezérelni:

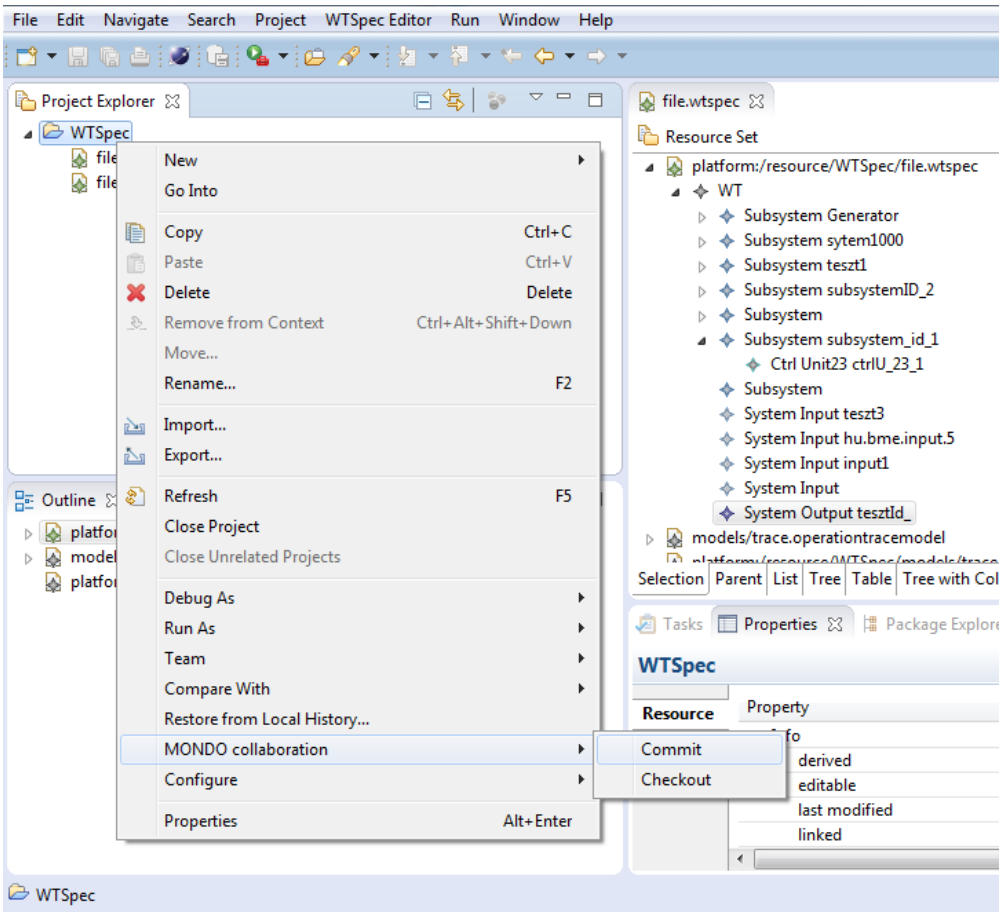
- **Commit**: feltölti helyi módosításokat a verziókövető szerverre.
- **Checkout**: lekéri a legfrissebb állapotát a modellnek a verziókövető szerverről.

A vezérlő felületet a projekten történő jobb egérgombbal történő kattintás hatására felugró menü kiegészítéseként valósítottam meg. (3.2. ábra)

A változtatások eléréséhez a *plugin.xml* fájl tartalmát kellett módosítani, ahol lehetőség van új menüelemek beregisztrálására. A menüpontokhoz kötött osztályoknak meg kell valósítaniuk az *IObjectActionDelegate* interfészt, melynek *run()* metódusában kerül lehetőség a menüpontra történő kattintás lekezelésére. A vezérlők feladata elindítani a jo-

gosultság ellenőrzést, amennyiben megfelelőnek találják az elvégzett módosításokat, kérelmeket a szabályzatokkal, abban az esetben meghívják MONDO kollaborációs kliens oldali interfész megfelelő metódusait.

A szerkesztői rendszeren a kollaborációs funkciókat csak hitelesített és azonosítót felhasználók vehetik igénybe. A felhasználók hitelesítése nem témája a dolgozatnak, viszont a hitelesítés és azonosítás eredményét felhasználja, innen ismeri a plug-in a felhasználót.



3.2. ábra A modell szerkesztő felhasználó felületének kiegészítése a kollaborációs vezérlőkkel

A felhasználói felületen a menüpontok létrehozása az alábbi kódrészlettel történik.

```
<extension
     point="org.eclipse.ui.popupMenus">
      <objectContribution
       id="ikerlanEMF.editor.contribution1"
       objectClass="org.eclipse.core.resources.IProject">
           <menu
                 id="ikerlanEMF.editor.menu1"
                 label="MONDO collaboration"
                 path="additions">
              <separator
                    name="group1">
              </separator>
           </menu>
           <action

class="eu.mondo.collaboration.accessprotocol.actions.CheckoutAction"
                 enablesFor="1"
                 id="ikerlanEMF.editor.newAction"
                 label="Checkout"
                 menubarPath="ikerlanEMF.editor.menu1/group1">
           </action>
           <action

class="eu.mondo.collaboration.accessprotocol.actions.CommitAction"
                 enablesFor="1"
                 id="ikerlanEMF.editor.commitAction"
                 label="Commit"
                 menubarPath="ikerlanEMF.editor.menu1/group1">
           </action>
      </objectContribution>
</extension>
```

A demonstrációs plug-in implementál egy osztályt, *UserDataBase*, ami tartalmazza a rendszer által ismert felhasználókat és azok szerepköreit. A plug-in felhasználásakor ezt át kell írni, hogy megfeleljen a produkciós környezetben használt felhasználó kezelő rendszernek. Az osztály egyetlen jelentős metódusa egy keresés, a *getUser(String userId)*, ami visszatér egy *User* objektummal a felhasználói adatbázisból felhasználói azonosító alapján. A felhasználói adatbázisban ismertek a felhasználókhoz rendelt szerepkörök, így a visszatért objektum már rendelkezik ezzel az adattal.

A fejlesztői rendszer aktuálisan használó felhasználó beállítása egy konfigurációs fájlban található. A plug-in gyökér mappájában található *config.properties* fájl tartalmazza melyik felhasználó nevében fut a fejlesztői környezet. A hozzáférési jogosultság lekérésekor ennek a felhasználónak kéri le a szerepköreit a plug-in, majd ennek a felhasználónak a nevében ellenőrzi a jogosultságot a házirenddel.

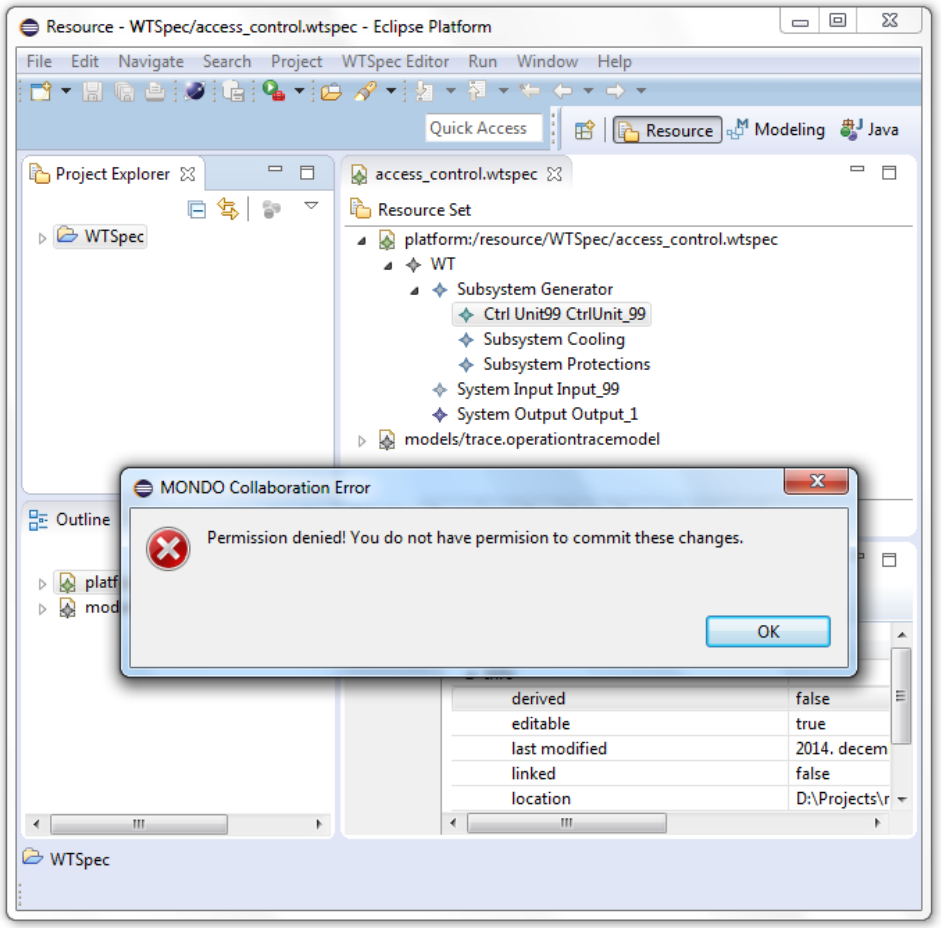
3.8. MONDO kollaboráció kliens oldali interfész

Az interfész kettő publikus metódussal rendelkezik, melyek logikai jelentéstartalma megegyezik a felhasználói felületen található kollaborációs vezérlőékkel. Mivel az interfész nyitott metódusokkal rendelkezik, nem feltételezhető, hogy a hozzá beérkező kérdések érvényesek-e, ezért ennél a pontnál is el kell végezni a felhasználó hozzáférési jogosultság ellenőrzését.

3.9. Kollaborációs modul tesztelése

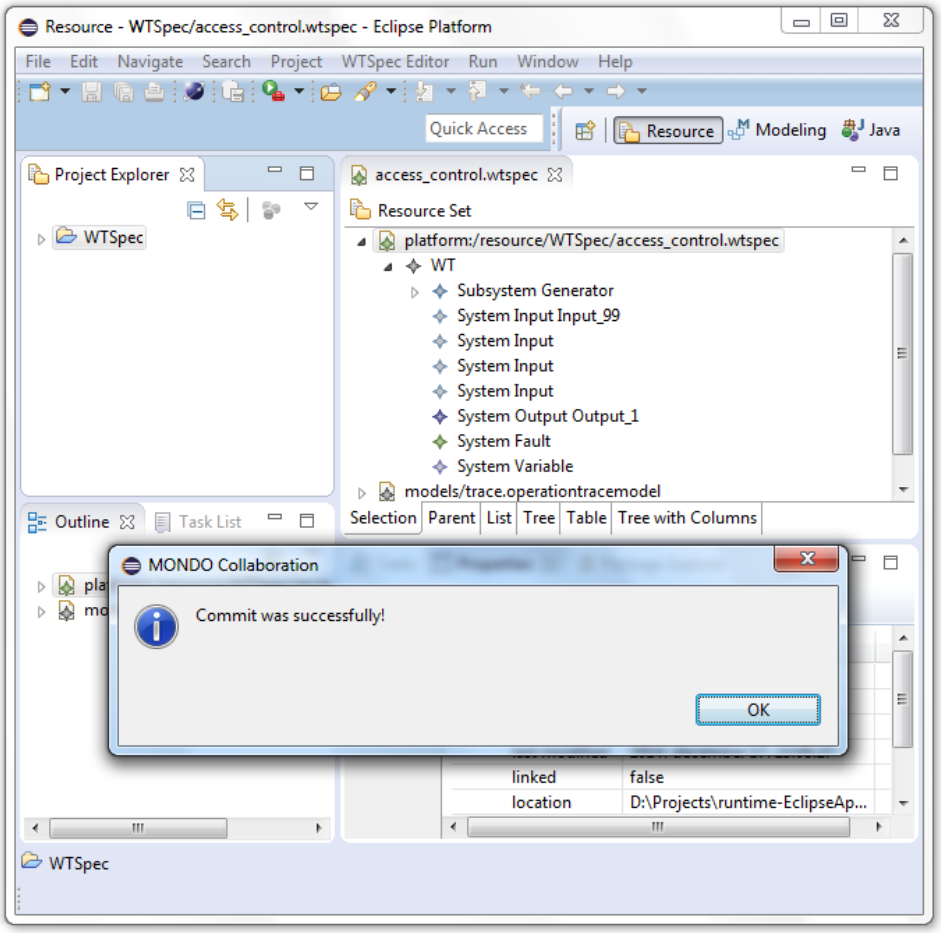
A plug-in tesztelésére a dolgozat bevezetőjében bemutatott esettanulmány példáját használom. A rendszer számára két felhasználó: *Alice* és *Bob*. Alice szerepköre az editor, míg Bob a designer szerepkörbe tartozik. Az editorok számára létezik az a megkötés, hogy nem dolgozhatnak *SystemInput* és *SystemOutput* elemekkel.

Alice nevében lefuttatva az első eredmény sikertelen, hiszen a modellhez hozzáadásra került új *SystemInput* elem.



3.3. ábra Alice sikertelen commit művelete

Míg ezzel szemben Alice második művelete, melyben nem ad a modellhez *SystemInput* vagy *SystemOutput* elemet engedélyezésre kerül. Ugyanígy Bob bármilyen nemű művelete sikeresen lefut, mivel Bob olyan felszanálói csoporthoz tartozik, akik számára nincs ilyen jellegű megkötés.



3.4. ábra Bob által indított commit sikeresen lefutott.

4. Összefoglalás

A dolgozatban kifejtett és lefejlesztett offline kollaborációt támogató Eclipse plug-in megítélése kutatási demonstrációs projektként értékelhető ki, mint sem önállóan működőképes szoftverként. A lefejlesztett program segítségével bemutatok egy lehetséges megoldási irányvonalát a hozzáférési jogosultságok lekérdezésének kollaborációs környezetben. A félév során lehetséges technológiákat kerestem a probléma megoldására és a leghatékonyabbakat kiválasztva készítettem el a jelen szoftvert. Mivel a dolgozatomban kifejezetten a hozzáférési jogosultságok lehetséges ellenőrzésére fókuszáltam, a szoftvernek nem készültek el olyan komponensei, amik nélkül demonstrációként megállja a helyét, viszont produkciós környezetben nem lehetne felhasználni. Ezeket részletesebben kifejtem a későbbi fejezetben.

A modell alapú jogosultság ellenőrzésre az XACML alapú megoldást megvalósíthatónak tartom, amit az éles szoftver esetén is ajánlatos lehet felhasználni. A nyelv által adott rugalmasság és a kiegészíthetőségének köszönhetően a speciális igények kiszolgálására is alkalmas lehet. Viszont mindenképpen javasolt egy absztrakciós réteg biztosítása az XACML nyelvű erőforrások elrejtésére, mivel a nyelv nehezen olvasható, szerkeszthető. A dolgozatban használt ALFA plug-in egy jó alternatívát nyújt erre az absztrakcióra az erőforrások kézi szerkesztésénél. További előnye könnyű kiegészíthetősége. Viszont az XACML programok számára történő feldolgozására nehezen található jó létező megoldás. A dolgozatban használt Balana projektet találtam a legjobban felhasználhatónak, viszont használata közben bebizonyosodott, hogy jelentősen összetettebb feladatok ellátására nem alkalmas a szoftvercsomag. Negatívumként kiemelném a zárt szerkezetét, körülményesen valósítható csak meg az új attribútumok, kategóriák definiálása.

4.1. Továbbfejlesztési lehetőségek

A demonstrátor szoftver tartalmazza az ALFA házirend szerkesztőt, viszont további fejlesztést igényelne, míg ebből a szerkesztőből egy teljes funkcionalitású PAP modul válna. A szerkesztő például ismerhetné a fejlesztő rendszer ecore modelljét, hogy már a házirend írása közben fel lehessen használni a modell szerkezetét, például ne lehessen nem valódi attribútumokat kényszerként megadni a szabályoknál. Vagyis az ALFA által biztosított XACML validálás mellett egy ecore modell alapján történő validáció fontos igény lehet.

A felhasználó kezelés nem volt témája ennek a dolgozatnak, így egy egyszerű demonstráló segédosztályokkal működik a felhasználó azonosítás. Elképzelhetőnek és praktikusnak találom, a fejlesztő rendszert egy LDAP címtáras felhasználó kezelő rendszerrel való integrációt. Továbbá a plug-in részeként szükséges egy konfigurációs felület, ahogy a felhasználó bejelentkeztetése történne, hogy a plug-in innen ismerje az aktuális felhasználó azonosítóját.

A házirend definiálásánál lehetőséget kellene biztosítani a hierarchikus letiltáshoz, vagyis ha egy konténer elem szerkesztéséhez nem kap jogot a felhasználó, akkor a konténer elem gyerekeit se tudja szerkeszteni.

Továbbá szükségesnek tartom a *WorkspaceTracker* továbbfejlesztését olyan irányba, hogy eltárolja egy létrehozott elemről a létrehozója azonosítóját, így az objektumokhoz tartozna egy tulajdonos, aki alapértelmezetten mindig rendelkezne szerkesztési joggal, hacsak explicit letiltásra nem kerül. ALFA segítségével a tulajdonos keresés megoldható, viszont hiányzik az adatforrás ehhez.

A felhasználói felület javítása is hozzátartozik a továbbfejlesztési lehetőségekhez. A hiba detektálása után a problémás elemeke megjelenítésre kerülhet a *Problems* nézet felületén az Eclipse keretrendszer *Markup* jelöléseit használva.

Ábrák jegyzéke

Irodalomjegyzék

[1] Coyne, Ed, and Timothy R. Weil. *ABAC and RBAC: Scalable, Flexible, and Auditable Access Management.* IT Professional 15.3 (2013): 0014-16.

[2] Hu, Vincent C., et al. *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. NIST Special Publication 800 (2014): 162.

[3] Axiomatics honlapja, ABAC, http://www.axiomatics.com/attribute-based-access-control.html.

[4] OASIS. XACML, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[5] Axiomatics. XACML, http://www.axiomatics.com/pure-xacml.html.

[6] Axiomatics. *ALFA Plugin for Eclipse User's Guide* (2013).

[7] WSO2: Balana, https://github.com/wso2/balana.

[8] The Eclipse Fundation. http://www.eclipse.org/.

[9] The Eclipse Foundation. Eclipse modeling framework. http://www.eclipse.org/modeling/emf/.

[10] Eclipse, EMF Ecore dokumentáció, http://download.eclipse.org/modeling/emf/emf/javadoc/2.7.0/org/eclipse/emf/ecore/package-summary.html.

[11] Christian Krause, *Henshin Example: Bank Accounts*, https://www.eclipse.org/henshin/examples.php?example=bank.

[12] MONDO Project. http://www.mondo-project.org/.

[13] Vikár András, *Kollaboratív modellező keretrendszer fejlesztése* (2014)

Függelék

[1] ALFA kódból generált teljes XACML házirend:

```
<?xml version="1.0" encoding="UTF-8"?>
 <!--This file was generated by the ALFA Plugin for Eclipse from Axiomatics
AB (http://www.axiomatics.com).
 Any modification to this file will be lost upon recompilation of the source
ALFA file-->
<xacml3:Policy xmlns:xacml3="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17"
    PolicyId="http://axiomatics.com/alfa/identifier/documents.topLevel"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
algorithm:deny-overrides"
    Version="1.0">
    <xacml3:Description />
    <xacml3:PolicyDefaults>
        <xacml3:XPathVersion>http://www.w3.org/TR/1999/REC-xpath-
19991116</xacml3:XPathVersion>
    </xacml3:PolicyDefaults>
    <xacml3:Target>
        <xacml3:AnyOf>
            <xacml3:AllOf>
                <xacml3:Match
MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
                    <xacml3:AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">document</xacml3:Attribute
Value>
                    <xacml3:AttributeDesignator

AttributeId="http://example.com/xacml/attr/resource/type"
                        DataType="http://www.w3.org/2001/XMLSchema#string"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource"
                        MustBePresent="false"
                    />
                </xacml3:Match>
            </xacml3:AllOf>
        </xacml3:AnyOf>
    </xacml3:Target>
    <xacml3:Rule
            Effect="Permit"

RuleId="http://axiomatics.com/alfa/identifier/documents.topLevel.Id_23">
        <xacml3:Description />
        <xacml3:Target />
        <xacml3:Condition>
            <xacml3:Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
                <xacml3:Function
FunctionId="urn:oasis:names:tc:xacml:1.0:function:integer-greater-than-or-
equal"/>
                <xacml3:AttributeDesignator

AttributeId="http://example.com/xacml/attr/subject/clearance"
                    DataType="http://www.w3.org/2001/XMLSchema#integer"
```

```
                    Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject"
                    MustBePresent="false"
                />
                <xacml3:AttributeDesignator

AttributeId="http://example.com/xacml/attr/resource/classification"
                    DataType="http://www.w3.org/2001/XMLSchema#integer"
                    Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource"
                    MustBePresent="false"
                />
            </xacml3:Apply>
        </xacml3:Condition>
    </xacml3:Rule>
    <xacml3:Rule
            Effect="Deny"

RuleId="http://axiomatics.com/alfa/identifier/documents.topLevel.Id_24">
        <xacml3:Description />
        <xacml3:Target />
        <xacml3:Condition>
            <xacml3:Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
                <xacml3:Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of">
                    <xacml3:Function
FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
                    <xacml3:AttributeValue

DataType="http://www.w3.org/2001/XMLSchema#string">draft</xacml3:AttributeVal
ue>
                    <xacml3:AttributeDesignator

AttributeId="http://example.com/xacml/attr/resource/documentStatus"
                        DataType="http://www.w3.org/2001/XMLSchema#string"
                        Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource"
                        MustBePresent="false"
                    />
                </xacml3:Apply>
                <xacml3:Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:not" >
                    <xacml3:Apply
FunctionId="urn:oasis:names:tc:xacml:1.0:function:any-of-any">
                        <xacml3:Function
FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal"/>
                        <xacml3:AttributeDesignator

AttributeId="http://example.com/xacml/attr/resource/documentAuthor"

DataType="http://www.w3.org/2001/XMLSchema#string"
                            Category="urn:oasis:names:tc:xacml:3.0:attribute-
category:resource"
                            MustBePresent="false"
                        />
                        <xacml3:AttributeDesignator

AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
```

```
DataType="http://www.w3.org/2001/XMLSchema#string"
                              Category="urn:oasis:names:tc:xacml:1.0:subject-
category:access-subject"
                              MustBePresent="false"
                          />
                      </xacml3:Apply>
                  </xacml3:Apply>
              </xacml3:Apply>
          </xacml3:Condition>
    </xacml3:Rule>
</xacml3:Policy>
```