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

2015

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, 2015. 05. 24.

.....

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

1.1. A projekt háttere

A dolgozat elkészítésének idejére csatlakoztam a MIT tanszéken futó MONDO [1] Európai uniós projekthez, melyhez tartozó kutatásokat a Hibatűrő Rendszerek Kutatócsoport végez el. A projektben résztvevői között más európai egyetemek és ipari partner cégek szerepelnek. A partner cégek modellvezérelt fejlesztőrendszereket használnak, melyekben a komplex modellek nagy méretei skálázódási problémákat vetnek fel. A projekt célja fejlesztőeszközök skálázhatósági problémájára megoldás biztosítása. A projekt további célja, hogy megvalósulhasson egy *Online* és egy *Offline* kollaborációra alkalmas modellező fejlesztőrendszer.

A kollaboratív modellezés során a fejlesztők párhuzamosan dolgoznak ugyan azon a modellen. A közös munka biztosítására két megoldási architektúra létezik: az *Online* és az *Offline* kollaboráció. Az *Online* megoldás esetén a fejlesztők folyamatos kapcsolatban állnak a munkavégzés közben a modellel. Amennyiben módosítást végzenek rajta, a módosítás hatása azonnal észlelhető a többi aktív részvevő számára is. Az *Online* kollaboráció kiemelt kihívásai közé tartozik többek között az aktív felhasználók módosításainak konkurenciakezelése.

Míg az *Offline* megközelítés értelmében a munkavégzés ideje alatta a fejlesztőrendszer és közös modell között nincs kapcsolat. A munkavégzés előtt a fejlesztő letölti a modell aktuális állapotát a fejlesztőrendszerébe. A módosítások csak a modell lokális másolatán léteznek. A módosítások befejeztével a módosított modell feltöltésre a kerül a közös modell tárhelyére. Egy módosítást, akkor észlelhető a többi felhasználó számára, mikor azok is frissítik a lokális modelljeiket. Kiemelt probléma a modellen a változások nyomon követése és a modellek összefésülése. Illetve a modell aktuális állapotának nyilvántartása, mivel nem lehet engedélyezni azt az esetet, hogy egy régebbi elavult állapot felülírja a friss modellt, aminek hatására adatvesztés lépne fel. Ezt a fajta szemléletmódot követik az olyan széles körben elterjedt verziókövető rendszerek is, mint például a GIT és az SVN is.

A projekthez rajtam kívül több diáktársam is csatlakozott szakdolgozatuk elkészítése céljából, név szerint Kamrás Márton, Molnár Ákos, Tunner Márton és Vikár András. A fel-

adataink elkülönülnek egymástól, viszont előfordulnak bizonyos határterületek, ahol találkoznak ezek a feladatok, amennyiben a dolgozatomban érintek ilyen területet, azt jelzem. A dolgozatban bemutatott forráskódok megtalálhatóak a projekt számára fenntartott GitHub tárhelyen. [1]

1.2. Jogosultság kezelés kollaboratív rendszerekben

Egy kollaboratív rendszer számára elérhető információk és adatok között lehetnek olyanok, amelyeket a tartalmuk miatt védeni kell az illetéktelen hozzáféréstől. Illetve egy ilyen rendszer felhasználói nem feltétlen rendelkeznek azonos szerepkörökkel. Amennyiben mindenki hozzáférhet minden adathoz, az a rendszer átláthatatlanságához vezethet. A hatékony együttműködés eléréséhez érdemes megkötéseket bevezetni a rendszerbe Ilyen megkötés, ha szeparáljuk az erőforrásokhoz való hozzáféréseket. A jogosultságok kiosztásánál általában érdemes minden felhasználónak a számára szükséges minimális hozzáférési jogkört biztosítani.

További előnye a hozzáférési jogosultág kezelésnek, hogy egy adott felhasználói csoport elől elrejthetőek a számukra nem releváns egységek. Így egy komplex fejlesztői rendszer használatát lehet egyszerűsíteni a felhasználók egy csoportja számára.

A jogosultság kezelő szabályokból dolgozik, amik meghatározzák az alanyok és az erőforrások közötti jogosultsági viszonyt. A jogosultság kezelő rendszerek használhatóságának fontos kritériuma, hogy az általuk kezelt erőforrások legkisebb felbontása megegyezzen a kollaboratív rendszerével. Például egy modellvezérelt fejlesztői rendszer számára, ami a modell részegységeivel és azoknak az attribútumaival dolgozik, nem feltétlen kielégítő, ha a jogosultságok csak fájlokra határozhatóak meg, mivel előfordulhat, hogy a teljes modell egy fájlban található.

1.3. Feladat meghatározás

A dolgozat során az alábbi feladatokat végeztem el:

- Kliens oldali *Offline* hozzáférési jogosultság kezelő keretrendszer megtervezése és megvalósítása
- Lekérdezés alapú jogosultság ellenőrzés kialakítása, mellyel lehetőség nyílik nagy modelleken végzett műveletek alapján azonnal kiértékelni a hozzáférési szabályokat és ellenőrizni azok esetleges megsértését

A modell vezérelt fejlesztőrendszerek számára a hozzáférési jogosultságokat a modell elemeihez és attribútumaihoz szükséges kötni, emiatt áttekintettem a felhasználható attribútum alapú hozzáférési jogosultság leíró megoldásokat. Így a megvalósított jogosultság kezelő rendszerem egy sztenderd jogosultság leíró nyelv (XACML) alapján értékeli ki a modellen végzett műveleteket.

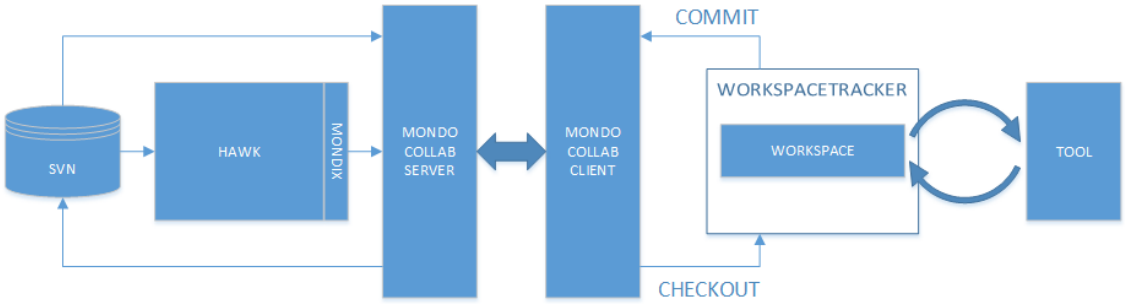
A jogosultság alapú ellenőrzéshez kialakítottam egy egyéni szabályleíró nyelvet és a hozzá tartozó szerkesztő felületet, mely pontosan illeszkedik a szükséges igényekhez.

1.4. Dolgozat felépítése

A dolgozatot a MONDO projekt *Offline* kollaborációs rendszerének struktúráját mutatom be. Ezt követi a projektben résztvevő ipari partnerek által leírt, a dolgozatom témájához legjobban illeszkedő, motivációs esettanulmány kifejtése. A *negyedik fejezetben* összefoglalom a hozzáférési jogosultság kezelési megoldásokat és azok felhasználásukra néhány példát állítok. Az *ötödik fejezetben* felsorolom és röviden bemutatom a dolgozatban szereplő technológiákat. A *hatodik* és *hetedik fejezetekben* bemutatom a feladatom által előírt általam tervezett és fejlesztett keretrendszer működését. A dolgozat zárásaként összegzem munkámat és megfogalmazom a lehetséges továbbfejlesztési pontjait.

2. MONDO kollaborációs rendszer

Az alábbi fejezetben röviden bemutatom a projekt *Offline* kollaborációs rendszer szerkezetét



2.1. ábra MONDO projekt *Offline* kollaboráció egységei

Az **2.1. 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.

3. 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 spanyol cég szélerőműveket tervez, a tervezéshez egy modell alapú Eclipse fejlesztőrendszert használnak.

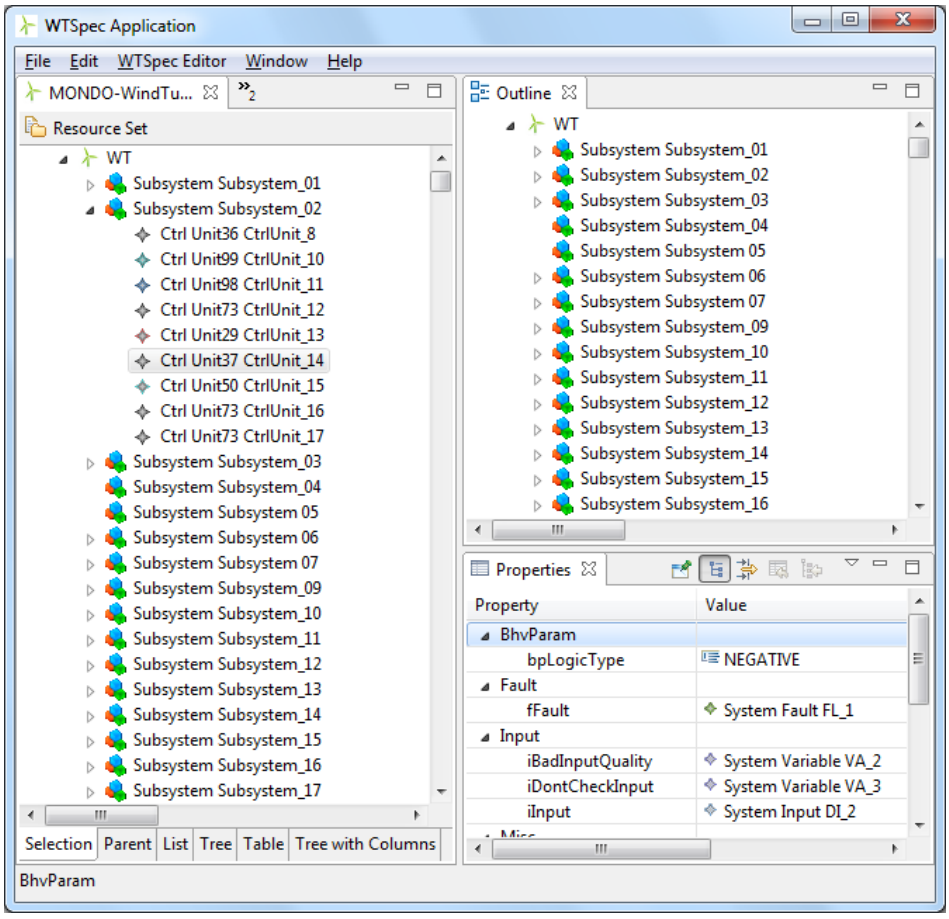
A modellező rendszerük meta-modelljében az alábbi objektumok találhatóak meg:

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

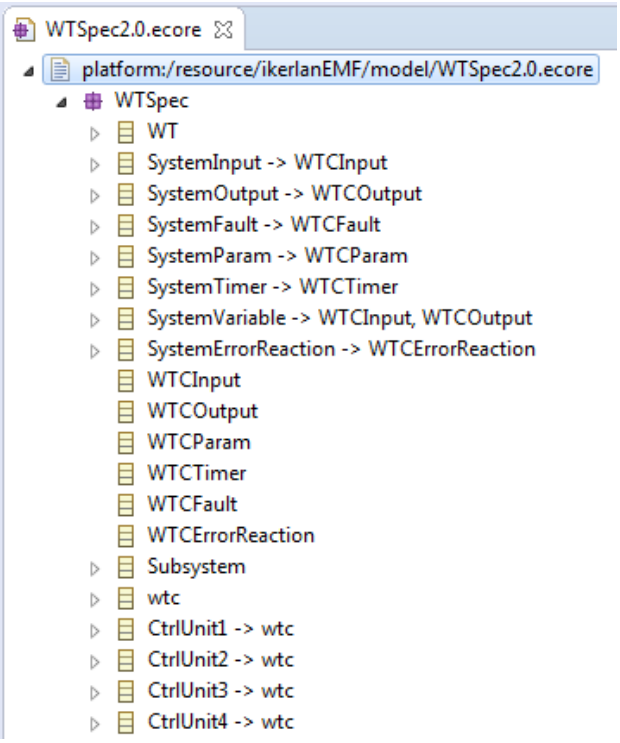
Minden meta-modell egy WT típusú gyökérelemmel kell kezdődnie. Az alrendszer elemek tetszőleges mélységben tartalmazhatnak további alrendszer elemeket. Rendszer elemeket csak a WT gyökér gyerek elemeiként lehet hozzáadni a modellhez. A szabályzó elemek hivatkozást tartalmaznak rendszer elemekre és csak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

A modellező eszközben az alábbi műveletek lehetségesek:

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



3.1. ábra IkerLan modellező program felhasználó felülete működés közben



3.2. ábra IkerLan modellező eszköz által használt Ecore modell részlete

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)

4. 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.

4.1. 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. [2][3]

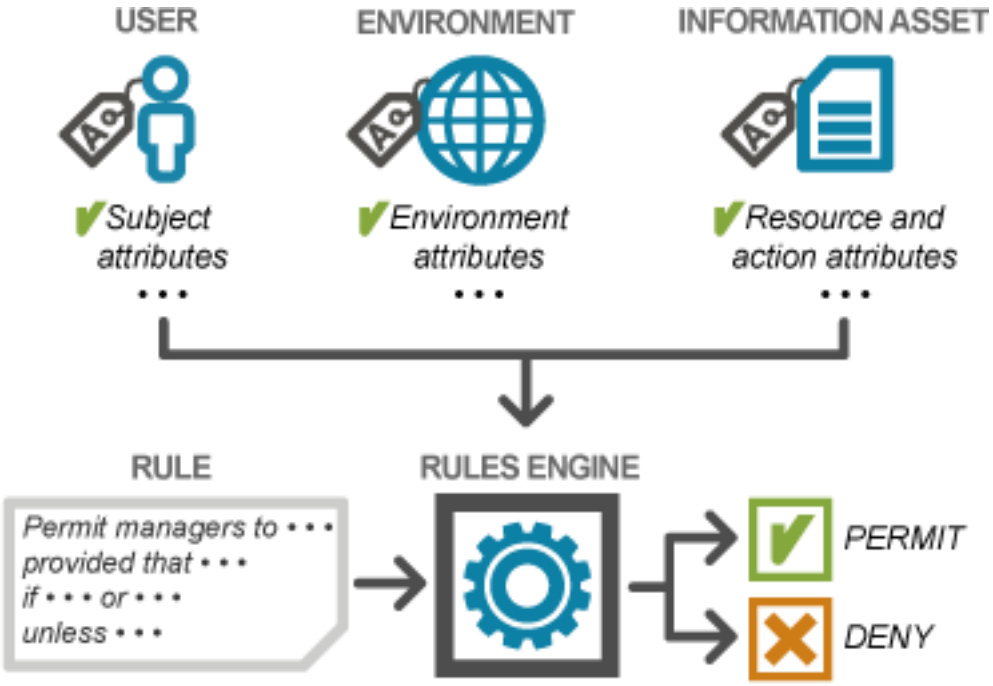
4.2. Attribútum alapú hozzáférési jogosultság kezelés (ABAC)

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. [3]

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). [2]



4.1. ábra Attribútum alapú hozzáférési jogosultság ellenőrző működését bemutató sematikus ábra [4]

4.3. Hozzáférési jogosultságok felhasználása

Verziókövető rendszerek

A verziókövető rendszerek a tárolandó adatokat egymástól logikailag és fizikailag is egymástól szeparált *repository*-kba szervezi. Manapság a két legelterjedtebb verziókövető rendszer jogosultság kezelését vizsgáltam, ezek a Git és az SVN. Mind a Git és az SVN rendszerek támogatják a hozzáférési jogosultságok meghatározását, ami külön-külön *repository*-kra vonatkoztatható. [5][6]

Viszont jelentős eltérés, hogy míg az SVN-ben van lehetőség a *repository*-n belül fájl szinten is jogosultság meghatározására, a Git ezt nem támogatja.

Adatbázisok

A relációs adatbázisok az információt táblákba rendezve tárolják, melyek oszlopokból épülnek fel. Relációs adatbázisok közül a MySQL és az Oracle rendszereit vizsgáltam. A hozzáférések meghatározásának legkisebb egysége mindkét rendszer az oszlop. E mellet

többek között lehetőség van tábla, illetve adatbázis szinten definiálni a felhasználók hozzáféréseit. Legnagyobb különbség a két rendszer között az, hogy míg az Oracle támogatja a felhasználók jogosultságát szerepkörökhöz kötni, addig MySQL esetén minden felhasználónak egyesével kell definiálni a jogköreit. [7][8]

Az adatbázisok egy másik megközelítése az úgynevezett *Triplestore*-ok. Ellentétben a relációs adatbázisokkal egy *Triplestore* állításokat tárol, melyek az alábbi adat hármasból épülnek fel: alany, tulajdonság és tárgy. Ennek a megközelítésnek köszönhetően a lehetőség nyílik egy dinamikus attribútum alapú hozzáférés jogosultság ellenőrzés megvalósítására, melyeknek a legkisebb hozzáférési egysége maga egy adathármas. [9][10] A különböző Triplestore implementációk nagyban eltérnek egymástól, vannak olyanok, amik megvalósítják az ilyen fajta hozzáférés kezelést, (AllegroGraph [11]), mások pedig nem (Jena TDB [12]).

Ezt a típusú alacsony szintű adat alapú hozzáférés jogosultság kezelést valósítja meg az Oracle Label Security egység relációs adatbázisokhoz, ami sor szintű hozzáférési szabályok definiálására ad lehetőséget. [13]

5. Technológiai áttekintés

Az alábbi fejezetben összefoglalom a dolgozat elkészítése közben felhasznált technológiákat.

5.1. XACML

Az XACML [14] (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:

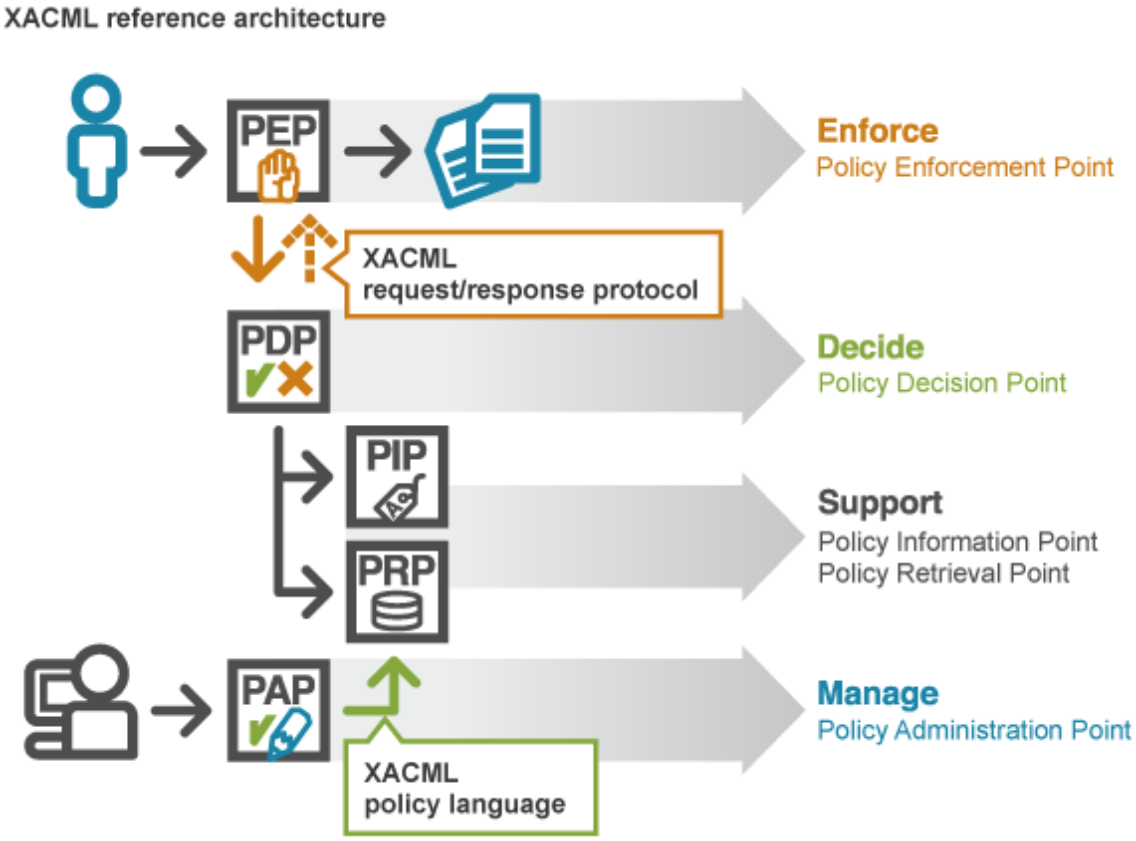
- **Házirend** (Policy): szabályzat, ami meghatározza a hozzáférési jogosultságokat.
- **Kérés** (Request): a felhasználó által küldött kérés tartalmazza műveletet jellemző összes attribútumot.
- **Válasz** (Response): jogosultság ellenőrző által kibocsátott válasz, melyben jelzi, hogy elfogadta, vagy 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.



5.1. ábra Hozzáférési jogosultság ellenőrző rendszer sematikus ábrája, mely XACML alapú házirendet használ. Az ábrán látható, hogy melyik modul melyik másik modult szolgálja ki információval. [15]

5.2. 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.

5.2.1. Felépítése

Az ALFA plugin tartalmaz egy Xtext alapú szerkesztőt, melynek előnye, hogy rendelkezik automatikus kiegészítő funkcióval és szerkesztés közben is ellenőrzi a kód érvényességét. A plugin 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 [16] 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.

[1] A teljes generált XACML állomány megtalálható a Függelékben [2]

Az ALFA nyelv az alábbi elemekből épül fel:

- **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.

5.2.2. Használata

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

Amint a plugin észleli az alfa kiterjesztésű fájl megnyitását, hozzáadja a projekthez az Xtext környezetet, ez gondoskodik a nyelvtan ellenőrzéséről. Ahhoz hogy felhasználhassuk a plugin 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.

5.3. 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 [17] 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.

5.4. Eclipse

Az Eclipse Foundation [18] 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 pluginek segítségével lehetséges.

5.5. Eclipse plugin fejlesztés

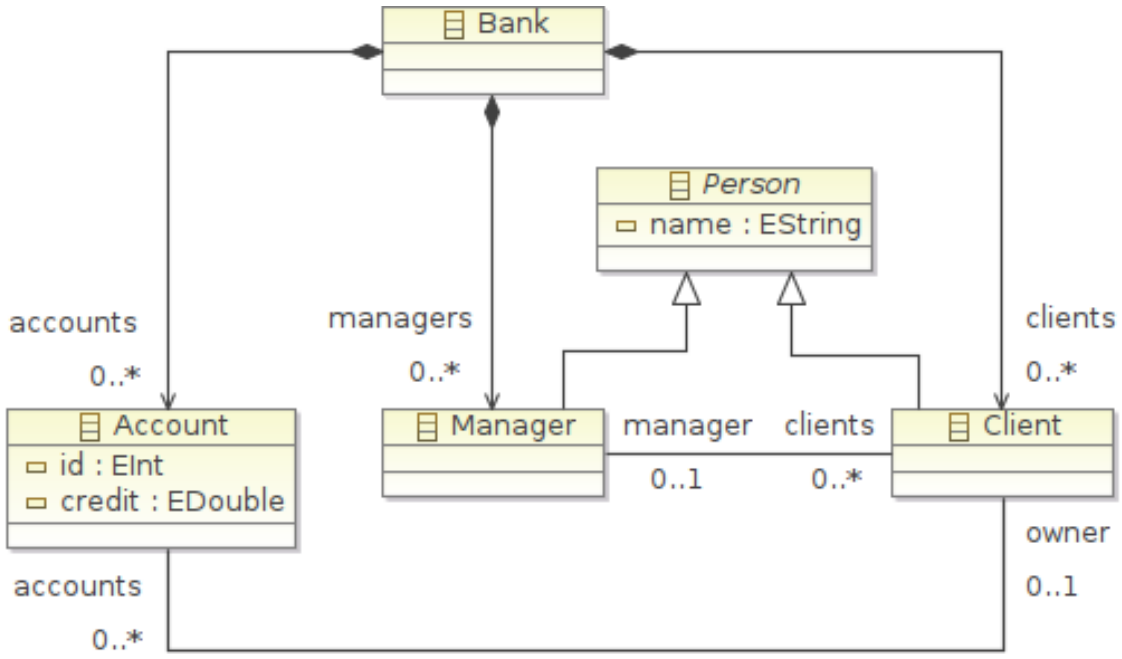
Két típusú plugin létezik az RCP (Rich Client Platform) és az RAP (Rich Ajax Platform). RCP segítségével egy független futtatható alkalmazást állítható elő. Az Eclipse egy plugin alapú keretrendszer, vagyis azoknak a funkcióknak a nagy része, amit plugin fejlesztésre használunk szintén egy plugin ként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

5.6. Eclipse Modeling Framework (EMF)

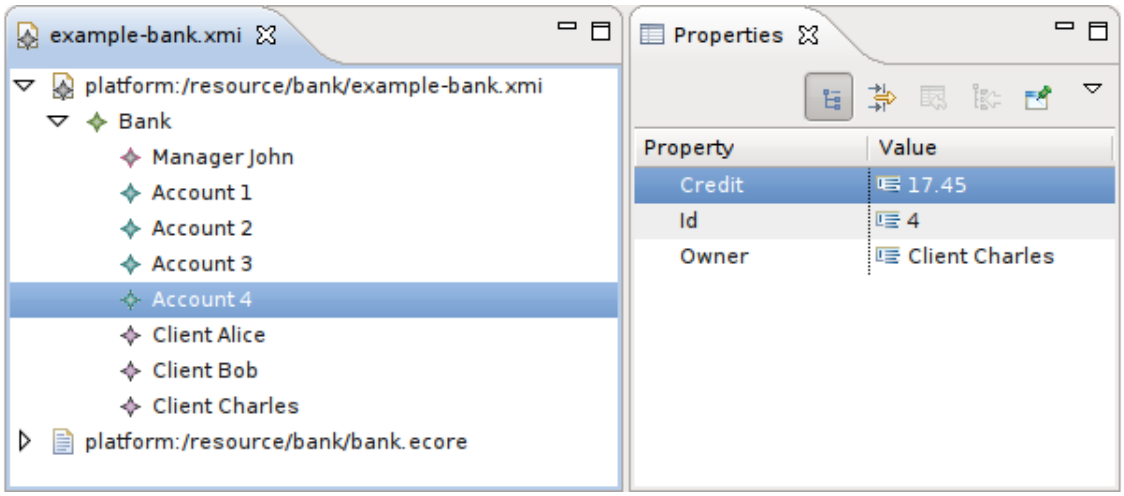
Az EMF keretrendszer lehetőséget biztosít modellező fejlesztőrendszerek készítésére. A modellek a *meta-modellen* alapulnak. A meta-modell az UML osztálydiagramjához hasonlóan egy olyan jellegű leírás, ami meghatározza, hogy milyen típusú modell elemek hol tárolódhatnak, illetve melyik entitás mely más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

modell elkészítését követően a keretrendszer elvégzi a meta-modellt leíró osztályok generálását. [19]

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



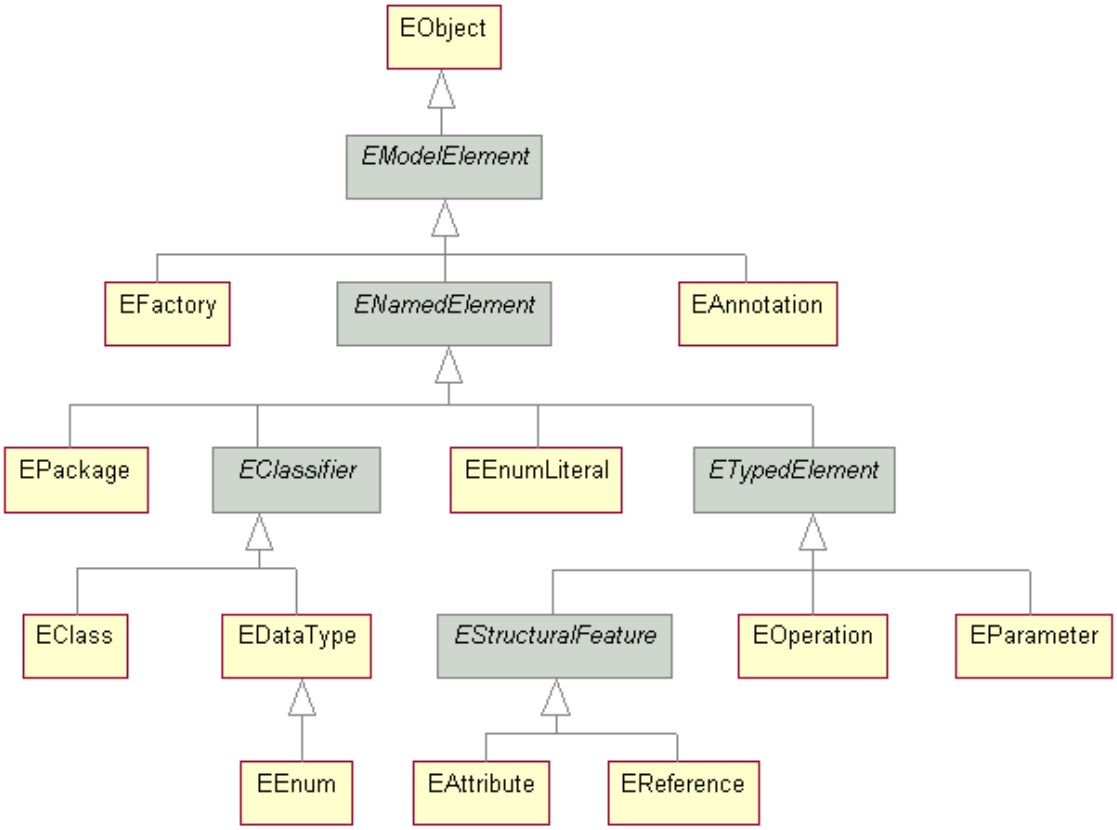
5.2. ábra Ecore diagram grafikus szerkesztőfelülete és a diagram vizualizációja [22]



5.3. 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 plugin megvalósítása.

Ecore modell készítéséhez az alábbi osztályok állnak rendelkezésre:



5.4. ábra Eclipse EMF Ecore modellt felépítő osztályok hierarchiája[20]

5.7. EMF-IncQuery

Az EMF-IncQuery [21] egy nyílt forrású keretrendszer, mellyel deklaratív lekérdezések futtatása válik lehetségessé EMF modellek felett. A gráf mintaillesztésen alapuló keresési megoldás miatt a lekérdezések futtatása jól skálázódik nagyméretű modellek esetén is. A lekérdezések az *eiq* kiterjesztésű fájlokban található keresési mintákból állnak. Előnye, hogy kézi kódolás nélkül is felhasználhatók benne az érintett modell objektumai és attribútumai. A lekérdezések definiálásra kifejlesztett nyelvtan segítségével bonyolultabb minták és feltételrendszerek is megfogalmazhatóak viszonylag kevés kóddal.

5.8. Xtext és Xtend

Az Xtext keretrendszer segítségével egyéni nyelvtan és a hozzá tartozó szerkesztőfelület tervezésére válik lehetőség. A keretrendszerrel fejlesztett szerkesztőfelület beépíthető az Eclipse felületébe, továbbá rendelkezik futásidejű szintakszis ellenőrzéssel, kód színezéssel és automatikus kód kiegészítési tulajdonságokkal.

Az Xtext szerkesztőbe épülő Xtend keretrendszer egy Java alapú programozási nyelv, aminek segítségével az Xtext szerkesztőben írt egyedi nyelvtan alapján összeállított kódból tetszőleges kód generálható. Ehhez az Xtext projekt *generator* csomagjában található *IGenerator* interfészt megvalósító osztály *doGenerate()* metódus törzsét kell implementálni.

6. Hozzáférési jogosultság ellenőrzés XACML alapokon

6.1. 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.

6.2. Hozzáférési szabályzat

6.2.1. 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 plugint.

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 attribútumok 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.

6.2.2. 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 az *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 szabadon 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"
                }
            }
        }
    }
}
```

6.3. 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 [23]*. A *WokspaceTracker* a projekten dolgozó egyik hallgatótársam munkája, jelen alfejezetben összefoglalom a modul működését, mivel a hozzáférési jogosultság ellenőrző plugin elengedhetetlen részét képezi ez a modul.

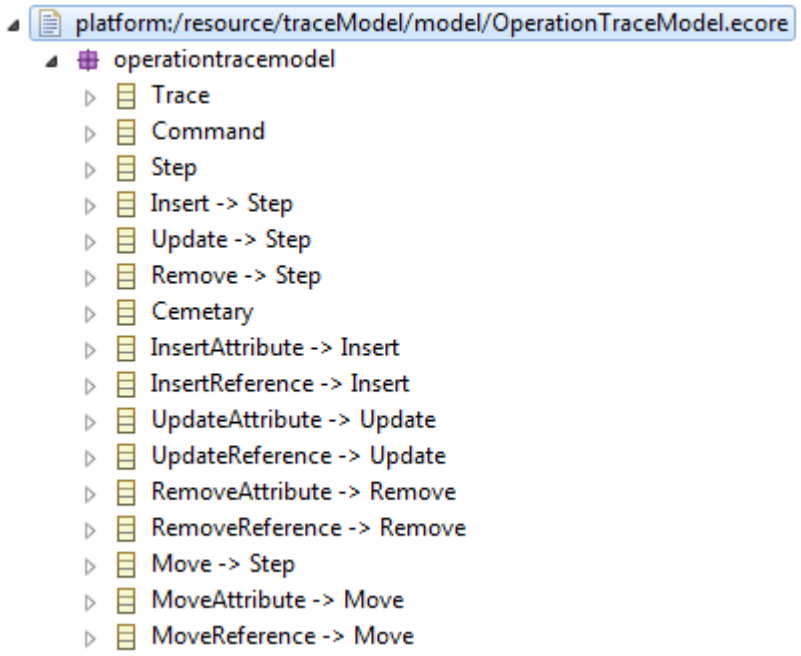
A *WorkspaceTracker* feliratkozást kezdeményez az Eclipse keretrendszer forrásfájlt nyomon követő eseményekre. Ez alapján a modul mindig értesül amennyiben változás jelent meg a beregisztrált forrásfájlban. A modul detektálja az egyes lépéseket, megállapítja azokról, hogy milyen típusú módosítást jelent a forrásmodellen. A sikeres beazonosítás után eltárolódik egy *operationaltracemodel* (6.1. 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.



6.1. 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.

6.4. 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();
}
```

6.5. 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>
```

6.6. Kérelem feldolgozása

A kérelem feldolgozását a PDP modul végzi. A pluginben lévő Balana alapú PDP modul használatához szükséges metódusok a *PDPHelper* osztályban találhatóak. A plugin 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 plugin 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));
```

6.7. Felhasználói felület

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

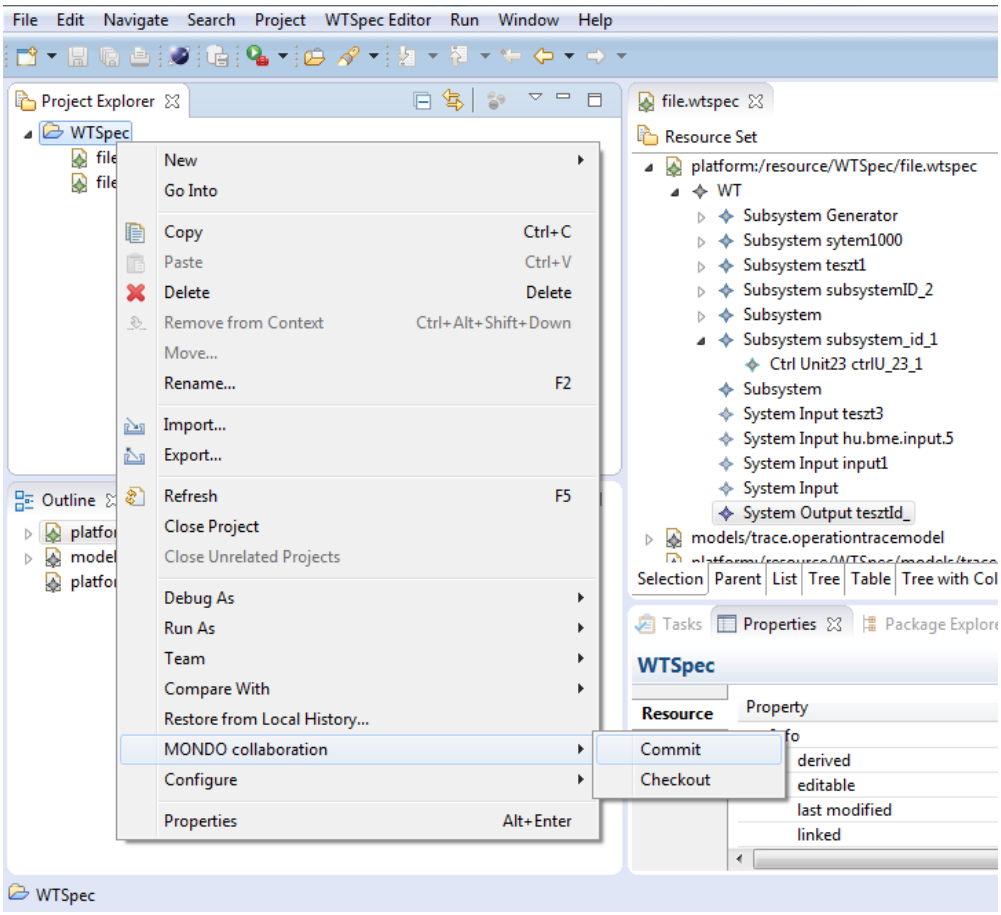
- **Commit**: feltölti helyi módosításokat a verziókövető szerverre.
- **Checkout**: lekéri a legfrissebb állapotát a modellnek a verziókövető szerverről.

A vezérlő felületet a projekten történő jobb egérgombbal történő kattintás hatására felugró menü kiegészítéseként valósítottam meg. (6.2. 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 plugin a felhasználót.



6.2. 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 plugin implementál egy osztályt, *UserDataBase*, ami tartalmazza a rendszer által ismert felhasználókat és azok szerepköreit. A plugin 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 plugin 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 plugin, majd ennek a felhasználónak a nevében ellenőrzi a jogosultságot a házirenddel.

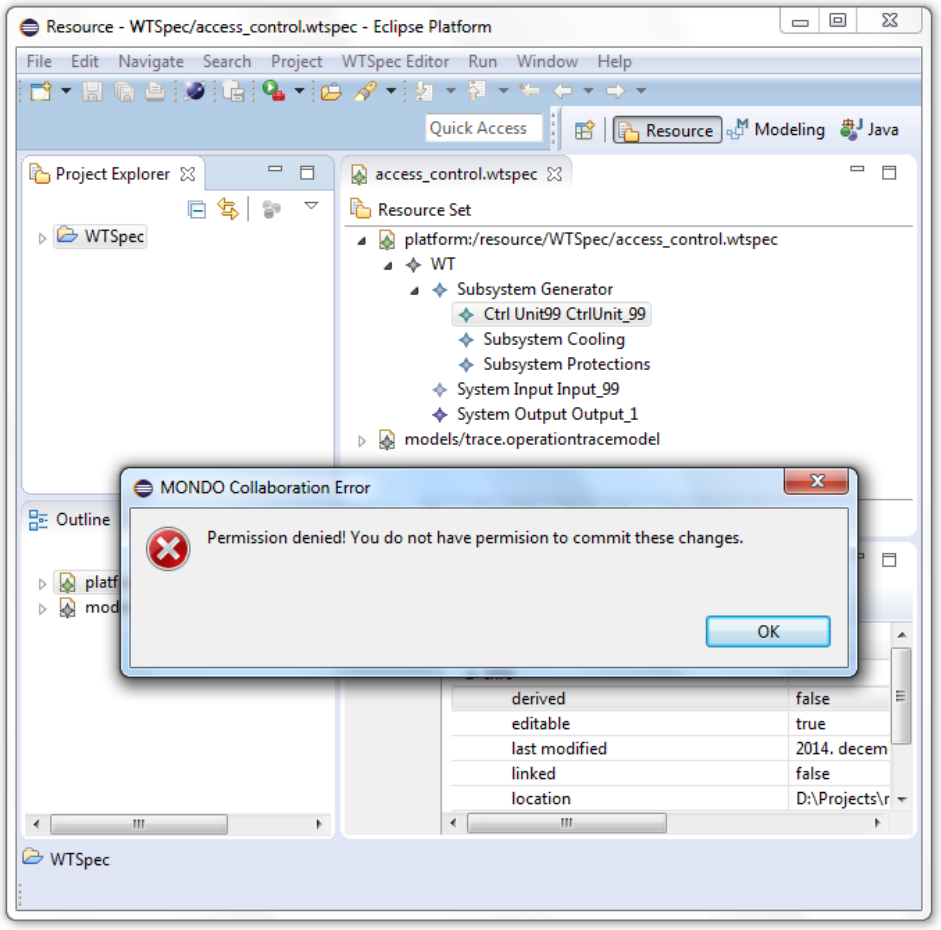
6.8. MONDO kollaboráció kliens oldali interfész

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.

6.9. Kollaborációs modul tesztelése

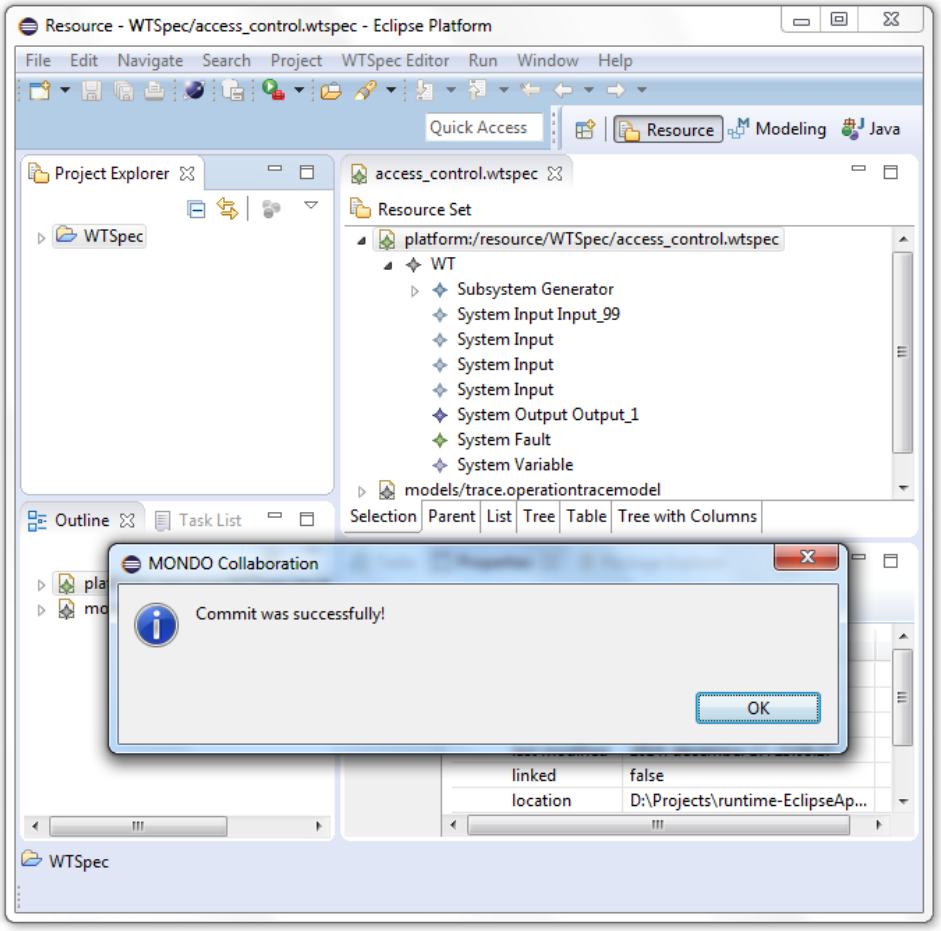
A plugin tesztelésére a dolgozat bevezetőjében bemutatott esettanulmány példáját használom. A rendszer számára két felhasználó: Alice és Bob. Alice szerepköre az editor, míg Bob a designer szerepkörbe tartozik. Az editorok számára létezik az a megkötés, hogy nem dolgozhatnak *SystemInput* és *SystemOutput*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

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



6.3. 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.



6.4. ábra Bob által indított commit sikeresen lefutott.

7. Hozzáférési jogosultság ellenőrzés INCQUERY lekérdezések alapján

7.1. Bevezetés

Az XACML alapú megközelít zárt rendszere miatt más megoldási alternatíva keresése révén a tanszéki kutatócsoporthoz által fejlesztett EMF-INCQUERY keretrendszerre esett a választás. A keretrendszer lehetőséget nyújt hatékony mintakeresésre nagy EMF modellek felett is. Ennek értelmében a jogosultság ellenőrzést egy olyan rendszerrel terveztem megoldani, ami kihasználja az EMF-INCQUERY ezen előnyét, viszont megoldható minden olyan eset is, ami a korábbi fejezetben bemutatott ALFA és XACML eszközökkel elérhető.

7.2. Tervezői döntések

Az EMF-INCQUERY mintákkal jól meg lehet határozni a komplex modell egyes részegységeit, melyek szerkesztését tiltani, vagy engedélyezni lehet a felhasználók számára. Erre lehetőséget biztosít a keretrendszer saját szerkesztő felülete, ahol a hozzáférési szabályhoz szükséges keresési mintákat *eiq* kiterjesztésű fájlokba gyűjthetőek.

Egy keresi minta összeköthető egy engedéllyel, ezzel egy szabályozási lehetőséget kapunk. Amennyiben a modellen lefutatott keresési minta találatot jelez, akkor a modellen tiltott vagy engedélyezett elem található. Az ilyen szabályok szabály gyűjteménybe szedhetőek, melyhez további tulajdonságok tartozhatnak, mint például milyen engedély dominájon engedély ütközés esetén, továbbá, hogy milyen felhasználókra vonatkoznak a szabályok.

A hozzáférési szabályok szerkesztéséhez létrehoztam egy Xtext szerkesztő felületet, ami a bevitt kódot feldolgozza és átalakítja egy JSON formátumú struktúrára. A szabály feldolgozó egység a JSON fájlból olvassa fel a szabályrendszert. A szabálystruktúra köztes sztenderd formátumú tárolását a rendszer moduláris szerkezetének megtartása érdekében választottam.

7.3. Hozzáférési szabályzat

Az Xtext keretrendszerrel létrehozott nyelvtan az alábbi elemekből épül fel: szabály és szabályok összerendelése. Egy szabályt meghatározza az egyedi azonosítója, az EMF-INCQUERY lekérdezéseket tartalmazó fájl elérési útvonala, a fájlban található minta azonosítója és a szabály engedélye, ami vagy engedélyezés, vagy tiltás. A nyelvtanban szabályt a *Policy* kifejezéssel lehet definiálni.

A szabályokat célponthoz lehet rendelni, ahhoz, vagy azokhoz a felhasználókhoz, akikre érvényesek a szabályok. Egy összerendelésnél definiálni kell a célpont típusát, ami vagy egy felhasználói csoport, vagy egyedi felhasználó lehet, valamint a célpont egyedi azonosítóját. Egy összerendelés tartalmazhat több szabályt is, ezért meg kell határozni egy összerendelésnél, hogy melyik típusú engedély dominál jon, valamint az alapértelmezett engedély típusát, amivel a hozzáférés jogosultság ellenőrző visszatér, ha nem talál eredményt. A szabályok célpontokhoz történő rendelése az *Association* kulcsszóval végezhető el.

Az előre rögzített konstans értékeket, mint például a felhasználói célpont típus választóját, vagy az engedély típusát definiáltam a nyelvtanban, így a szerkesztői felület segítséget nyújthat a szabályrendszert összeállító felhasználónak az automatikus kiegészítés funkcióval.

A nyelvtan szerkesztő használatához a szabályleíró fájlokat *policy* fájlkiterjesztéssel kell rendelkezniük. Ha a szabályrendszert tartalmazó projekt nem tartalmazza az Xtext környzetet, akkor a fejlesztőrendszer automatikusan felajánlja ennek elvégzését, mihelyst létrejött a *policy* típusú fájl.

Az következő példakódok bemutatják a nyelvtan használatát. A példa szabály megtiltja a *designer* típusú csoportba tartozó felhasználóknak a *SystemInput*, *SystemOutput* és a *CtrlUnit99* típusú elemek létrehozását, módosítását és törlését. Emellett definiál egy egyedi felhasználóra vonatkoztatott szabályt, ami engedélyezi azoknak a *CtrlUnit99* elemekkel történő műveleteket, amik a *Generator* azonosítóval rendelkező alrendszerben találhatóak.

Első megközelítésben definiálni kell azokat a lekérdezéseket, amikkel megtalálhatóak ezek az elemek a modellben, ez az alábbi EMF-INCQUERY lekérdezésekkel valósíthatóak meg, amiket a *quieries.eiq* fájlban definiálok.

```
package wtspec

import "http://www.eclipse.org/emf/2002/Ecore"
import "http://WTSpec/2.0";

pattern systemInput(I : SystemInput, n: EString) {
	WT.itsInputs(_WT, I);
	SystemInput.sysId(I,n);
}

pattern systemOutput(O : SystemOutput, n: EString) {
	WT.itsOutputs(_WT, O);
	SystemOutput.sysId(O,n);
}

pattern ctrlUnit99(S : Subsystem, s_id : EString, WTC : CtrlUnit99, c_id :
EString) {
	Subsystem.itsWTCs(S, WTC);
	CtrlUnit99(WTC);
	Subsystem.sysId(S,s_id);
	wtc.sysId(WTC, c_id);
}

pattern subSystemGenerator_CtrUnit99(S : Subsystem, s_id : EString, WTC :
CtrlUnit99, c_id : EString){
	Subsystem.itsWTCs(S, WTC);
	CtrlUnit99(WTC);
	Subsystem.sysId(S,s_id);
	wtc.sysId(WTC, c_id);
	check(s_id == "Generator");
}
```

Látható, hogy beimportálható a célmodell, így a szerkesztő ismeri a modell szerkezetét és kódolás nélkül felhasználhatóak az abban szereplő elemek és attribútumok.

Következőekben definiálni kell a szabályokat, amik felhasználják ezeket a keresési mintákat, illetve szabályok összerendelését. A tulajdonságok definiálásának sorrendje nem meghatározott. Megfigyelhető, hogy az összerendelésnél az alapértelmezett engedélyt jelző *defaultPermission* tulajdonság nem kötelező, viszont amennyiben kitöltésre kerül, akkor abban az esetben, ha nincs illeszkedés egyik szabály mintára se, akkor ezzel az engedély típussal tér vissza az ellenőrző egység. A korábban definiált EMF-INCQUERY mintákra a hivatkozás az *eiq* fájl elérési útvonalával, illetve lekérdezések csomag azonosítójuk és a minta azonosítójának kombinációjával. A szabály összerendelésnél több szabályra történő hivatkozásnál vesszővel kell elválasztani azokat. Az egyszer már definiált szabály több összerendelésnél is felhasználható.

```
Policy systemInput {
	permission DENY;
	query
"d://Projects//MONDO//hu.bme.mit.ftsrg.mondo.accesscontrol.incquery//src//wts
pec//queries.eiq";
	pattern "wtspec.systemInput";
}
Policy systemOutput {
	permission DENY;
	query
"d://Projects//MONDO//hu.bme.mit.ftsrg.mondo.accesscontrol.incquery//src//wts
pec//queries.eiq";
	pattern "wtspec.systemOutput";
}
Policy ctrlUnit99 {
	permission DENY;
	query
"d://Projects//MONDO//hu.bme.mit.ftsrg.mondo.accesscontrol.incquery//src//wts
pec//queries.eiq";
	pattern "wtspec.ctrlUnit99";
}
Policy subSystemGenerator_CtrUnit99 {
	permission ALLOW;
	pattern "wtspec.subSystemGenerator_CtrUnit99";
	query
"d://Projects//MONDO//hu.bme.mit.ftsrg.mondo.accesscontrol.incquery//src//wts
pec//queries.eiq";
}

Association {
	defaultPermission ALLOW;
	override DENY;
	target GROUP."designer";
	policies systemInput, systemOutput, ctrlUnit99;
}
Association {
	override ALLOW;
	target USER."alice";
	policies subSystemGenerator_CtrUnit99;
}
```

A policy fájl mentésekor automatikusan lefut az Xtend keretrendszer kódgenerátora, ami előállítja a későbbiekben felhasználásra kerülő szabályrendszert leíró JSON formátumú fájlt. A szerkesztői projektben létrejön egy *src-gen* mappa, ebben található a generált fájl, aminek neve megegyezik a policy fájl nevével, csak a kiterjesztésükben térnek el.[5]

Mivel a szabályszerkesztő felület és a szabályt végrehajtó felület két külön komponens, az XACML féle megközelítéshez hasonlóan, előnyösnek tartottam, ha egy sztenderd adatstruktúrán keresztül képesek a kommunikálásra. Így mindkét komponens használhat már létező feldolgozó könyvtárakat.

[5] A generált fájl tartalma megtalálható a Függelékben[3]

7.4. Hozzáférési szabályzat feldolgozása

Az egyedi szabályzat leírás miatt egyedi feldolgozó modul fejlesztése is szükségessé vált, ami a *hu.bme.mit.ftsrg.mondo.accesscontrol.policyverification* csomagban található meg. A célponthoz rendelt szabályzat objektumot a *PolicySet* osztály implementálja, míg egy szabály objektumot a *Policy* osztály. Ezeknek az osztályoknak a leszármaztatottjai a **PolicySetQuery** és a **PolicyQuery** osztályok, melyek a szabályok kiértékeléséhez szükséges metódusokkal és lokális változókkal egészítik ki az alap objektumokat.

A szabályírás során egy JSON tömb generálódik a szabályrendszerekkel, viszont ez a tömb rendezetlenül tartalmazza a szabályokat. Ugyanabban a tömbben megjelenhet felhasználókra, illetve a csoportokra vonatkozó szabályrendszerek is. A könnyebb kiértékelés érdekében ezeket a szabályrendszereket szelektálva tárolja a rendszer. A szeparált tárolásra használható a **PolicySetFactory** osztály, ami egy adott típusú (felhasználói, vagy csoport szintű) szabályrendszereket tárol csak a kollekciójában. A tárolásra használt kollekció egy HashMap, aminek kulcsa a célpont azonosító, értéke pedig az adott célponthoz tartozó szabályok.

A szabályzatok feldolgozása a **PolicyVerifier** osztályban fut le, itt elkülönülnek a felhasználókra és a csoportokra vonatkozó szabályrendszerek. A szeparálás kiemelt oka, hogy a kiértékelés során, ha illeszkedik a felhasználóra szabály, akkor az felülírja a csoporttól örökölt jogosultságokat. A feldolgozás során konfigurációs fájlban beállított mappából felolvassa az összes JSON fájlt, amik a szabályrendszert tartalmazzák és a szerepkörnek megfelelő gyűjteménybe szelektálja annak tartalmát. Emiatt nem kell egy fájlba beletenni az összes szabályzatot a szerkesztőben, a fájlokra osztás segítheti az átlátható struktúra fenntartását. A PolicyVerifier osztálynak ismernie kell az aktuális felhasználó azonosítóját, illetve, a csoportjának azonosítóját, mivel a szabályrendszer kiértékelésénél csak az érintett szabályokkal dolgozik a rendszer.

A feldolgozó egységet a modell szerkesztő pluginbe történő integrálását az **AccessControlManager** osztály végzi el. Feladata a szerkesztő rendszerből kinyerni az aktuálisan szerkesztett modellt, illetve a felhasználó adatait.

7.5. Hozzáférési szabályzat kiértékelése

A szabályzat kiértékelése az EMF-INCQUERY minták találatán alapulnak. Viszont nem lehet egyszerűen azt figyelni, hogy van-e a mintának megfelelő találat, hiszen előfordulhat, hogy a modell már akkor tartalmazott olyan elemeket, amiknek szerkesztése az aktuális felhasználó számára tiltott, amikor a felhasználó még nem módosított semmit se. Ezért az aktuális mintákat inicializálni kell a szerkesztő indításakor, ennek eredménye eltárolásra kerül a **PolicyQuery** objektum **originalMatcher** tagváltozójában, amihez a későbbiekben hasonlítani fogom az aktuális találatokat.

A kiértékelés igénye a PolicyVerifier osztálytól indul, viszont a tényleges kiértékelést a szabályzat legalacsonyabb pontjai a PolicyQuery objektumok végzik el saját keresési mintájukat lefuttatásával. Első indításkor a minták inicializálásakor az aktuális felhasználót érintő összes minta lefuttatásra kerül. Az ez utáni futtatások sebességét tekintve gyorsabbak, egyrészt mivel a kiértékelésnél, csak addig futnak a mintakeresések, amíg szabályellenes találatot nem észlelnek. Másrészt az EMF-INCQUERY sajátossága, hogy a keresési mintákat és találatokat tárolja a gyorsító memóriájában, így az ismétlődően visszatérő minta futtatások sebessége közel konstans sebességűvé redukálódik és független a modell méretétől a *RETE* algoritmusnak köszönhetően [24].

A minta keresésnél, amennyiben az eredeti találati lista és az aktuális találati lista darabszáma eltér egymástól, akkor egyértelműen olyan módosítás történt, amit ki kell vizsgálni, mivel vagy törlés, vagy új elem hozzáadása történt. Viszont ha a listák darabszáma megegyezik, akkor muszáj végigellenőrizni a két lista azonosságát, hiszen vagy nem történt érdemleges módosítás, vagy valamelyik attribútum módosult egy a szabályzat számára kiemelt objektumnál. EMF-INCQUERY minta keresés futtatásához Java kódból szükséges implementálni egy keresési motort (*AdvancedIncQueryEngine*), a vizsgálat tárgyának számító EMF modellt, továbbá az aktuális mintához tartozó lekérdezés specifikációhoz (*IQuerySpecification*). A minta kiértékelését végző kódrészlet megtalálható a **PolicyQuery** osztály **executeQueryOnPattern()** metódusában.

A szabályok kiértékelése után következik a szabályrendszerek kiértékelése. Miután a szabályok visszajeleztek, hogy található-e a modellben olyan módosítás, amit keresnek a szabály minták, a PolicySetQuery objektum meghatározza az érintett szabályok engedélyei, az alapértelmezett és a domináns engedély tulajdonságok alapján az összesített engedélyt. Ezt a folyamatot mutatja be az alábbi kódrészlet:

```
private Permission executePolicySet(Resource resource,
          boolean initialization) throws IncQueryException {
     boolean permissionDiff = false;
     Permission executedPolicies = null;
     boolean matchFound = false;
     Iterator<PolicyQuery> policyIterator = policies.iterator();
     // A szabálygyűjtemény kiértékelése addig, amíg eltérést nem
     // tapasztalható az alapértelmezett jogosultsággal
     while ((initialization == true || permissionDiff == false)
               && policyIterator.hasNext()) {
          PolicyQuery policy = (PolicyQuery) policyIterator.next();
          boolean policyMatchFound = policy.executeQueryOnPattern(resource,
                    initialization);
          if (initialization == false && policyMatchFound) {
               matchFound = true;
               if (policy.getPermission() != defaultPermission) {
                    permissionDiff = true;
                    if (policy.getPermission() != executedPolicies) {
                         executedPolicies = policy.getPermission();
                    }
               }
          }
     }
     if (matchFound) {
          if (permissionDiff == true) {
               // Szabálygyűjteményben található tiltás és engedélyezés is
               // A jogosultság felülíró tulajdonság dönti el a jogosultság
               // kezelést
               return permissionOverride;
          } else {
               // Szabálygyűjtemény nem mutat eltérést az alapértelmezett
               // jogosultságtól
               return defaultPermission;
          }
     } else {
          // Nincs találat egy szabálynál sem
          return null;
     }
}
```

A szabályzat minta kiértékelések mindig csak az összes szabályzat egy részhalmazán futnak le, mivel csak azokra a szabályokra futtatjuk le az ellenőrzés, amelyek a célpontja illeszkedik az aktuális felhasználóra. Ezt a szabályzat kiértékelés lefutásra kerül mind a csoport szintű szabályokon, mind a felhasználói szabályokon külön-külön, amennyiben léteznek. Amennyiben a csoport szintű jogosultság ütközik a felhasználói jogosultsággal, akkor a felhasználói szintű a dominánsabb.

7.6. Modell változásának észlelése

Az EMF-INCQUERY használatával lehetőség nyílik arra, hogy a használt keresési mintákat beregisztráljuk és amennyiben a modell változtatásának hatására módosul ezeknek a mintáknak a találati listája, akkor egy esemény kezelőre feliratkoztatott egyéni metódus meghívásra kerül. Így a rendszer ad jelzést, hogy a jogosultság ellenőrző állapota elavulhatott. Amennyiben egy minta eredményhalmaza módosult, abban az esetben le kell futtatni a teljes kiértékelést, mivel a szabály eredmények dominanciája és a szerepkör szerinti szabályrendszerek nem azonos prioritása miatt csak akkor kaphat a rendszer pontos képet az aktuális jogosultságokról.

A feliratkozáshoz szükséges ismerni azokat a keresési mintákat, amiket felhasználunk a kereséshez. Ezt onnan lehet kinyerni, hogy a keresési mintához tartozó lekérdezés specifikációt be kell regisztrálni a keresési minták inicializálásakor egy nyilvántartó kollekcióba.

```
if(QuerySpecificationRegistry.getQuerySpecification(getPattern()) == null){
     QuerySpecificationRegistry.registerQuerySpecification(specification);
}
```

Később innen kiolvasható az összes szükséges egyedi keresési minta. A kigyűjtött mintákat hozzáadom egy UnionSet típusú kollekcióhoz, ami támogatja az elemeinek változásának hatására kiváltott eseményre a feliratkozást.

Viszont az eseménykezelő sajátosságai miatt a feliratkoztatott minták tulajdonsága illetve a változás tulajdonságai miatt előfordulhat, hogy a feliratkoztatott függvényünk ugyanarra a változásra többször is meghívásra kerül. A fölösleges futtatás okozta terhelés és az esetleges rossz végeredmények elkerülése miatt ezt figyelni kell a kiértékelés indítása előtt.

Ennek a problémának a megelőzésre végett bevezettem két állapotváltozót a rendszeremben. Az egyik állapotváltozó a jogosultság kezelő pluginben (**AccessControlManager**) kerül tárolásra és minden modell változásnál léptetés történik az állapotában. A másik állapotváltozó a kiértékelésért felelős **PolicyVerifier** osztályban kerül tárolásra, értéke pedig megegyezik a plugin állapotváltozó akkori értékével, amikor az a kiértékelést elindította. A plugin állapotváltozójának léptetéséhez külön feliratkoztattam a modell változásainak nyomon követésére, ami akkor is meghívásra kerül, ha nem a szabályzatok által figyelt módosítás történik. Ennek köszönhetően a plugin állapotváltozójával nem fordul-

hat elő az, hogy egy változás hatására Végül az EMF-INCQUERY által jelzett modellváltozáskor csak akkor futtatom le a teljes jogosultság kiértékelőt, ha a két komponens állapotváltozója nem egyezik meg.

```
AdvancedIncQueryEngine engine = AdvancedIncQueryEngine
          .createUnmanagedEngine(resource);
Set<IQuerySpecification<? extends IncQueryMatcher<? extends IPatternMatch>>>
specifications = QuerySpecificationRegistry
          .getContributedQuerySpecifications();
ArrayList<IObservableSet> observableSetList = new
ArrayList<IObservableSet>();

for (IQuerySpecification<? extends IncQueryMatcher<? extends IPatternMatch>>
s : specifications) {
     IObservableSet set = IncQueryObservables.observeMatchesAsSet(s
               .getMatcher(engine));
     observableSetList.add(set);
}

IObservableSet[] observabelArray = observableSetList
          .toArray(new IObservableSet[observableSetList.size()]);
UnionSet union = new UnionSet(observabelArray);
union.addSetChangeListener(new ISetChangeListener() {

     @Override
     public void handleSetChange(SetChangeEvent event) {
          // Állpotváltozók ellenőrzése
          if (verifier.getLastExecutionStep() == null
                    || verifier.getLastExecutionStep() != getChangeState())
{
               // Jogosultság kiértékelés elindítása
               verifier.printExecutedResult(getChangeState());
          } else {
               // Az állapotváltozók megegyeznek, ekkor nem kell semmit
               // se csinálni
          }
     }
});
```

A jogosultság ellenőrző plugint a szerkesztő rendszerhez a *WTSpecEditor* osztály *createModel()* metódusában kell hozzáadni, mint egyéni *EAdpater*.

8. Összefoglalás

A dolgozatban kifejtett és lefejlesztett offline kollaborációt támogató Eclipse plugin 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 plugin egy jó alternatívát nyújt erre az absztrakcióra az erőforrások kézi szerkesztésénél. További előnye viszonylag könnyű kiegészíthetősége. Viszont az XACML programok számára történő feldolgozására nehezen található kellően támogatott 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, valamint nem használható képzi kódolás nélkül. Mivel a rendszer nem ismeri a modell szerkezetét, ezért szükséges az elemek és paraméterek definícióit kézi kódolással megadni.

Ezzel ellentétben az EMF-INCQUERY által biztosított lehetőségek megoldást jelenthetnek az XACML alapú megközelítés gyenge pontjaira. A modell változás hatására automatikusan lefuttatott hozzáférési ellenőrzés nagyon felhasználó barát felületet biztosít, hiszen azonnali visszajelzést kap egy esetleges tiltott műveletről. A lekérdezések hatékony és gyors lefuttatása további erősége ennek a megoldásnak. A házirend szabályok definiálásra tervezett nyelvnél azt próbáltam elérni, hogy a szabályok definiálásához ne kelljen

komplex szerkezeteket definiálni, minél egyszerűbben lehessen megfogalmazni a szükséges engedélyezéseket vagy tiltásokat. Továbbá az EMF-INCQUERY lekérdezések definiálására használt nyelvtan elsajátítása után közel tetszőleges komplexitású házirendek írására ad lehetőséget a rendszer. Jóval egyszerűbben bonyolultabb szabályok definiálhatók, így mint az ALFA vagy XACML segítségével.

A dolgozat írásával egy problémára két különböző alternatívát mutattam be. Az első megoldással sztenderd létező elemeket felhasználhatóságát és testreszabhatóságát igyekeztem kideríteni. Míg a második megoldással egy teljesen egyedi rendszer alapjait fektettem le. Látható, hogy az egyedi megközelítéssel és az EMF-IncQuery keretrendszer rugalmasságának köszönhetően ez a megoldás tűnik kivitelezhetőnek és felhasználóbarátnak.

8.1. 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 plugin részeként szükséges egy konfigurációs felület, ahogy a felhasználó bejelentkeztetése történne, hogy a plugin innen ismerje az aktuális felhasználó azonosítóját. A modell egyszerűsítése végett azt feltételeztem, hogy egy felhasználó csak egy csoporthoz tartozhat, viszont egy valós rendszerben ez nem feltétlen elképzelhető. Emiatt a jogosultság ellenőrzés kiegészítésre szorul, a szerepkörök konkurencia kezelésével.

Az XACML alapú 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. Erre alternatívát biztosít az EMF-INCQUERY lekérdezéseken alapú megoldás.

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.

Jelen állapotban az EMF-INCQUERY mintákat felhasználó szabály definiálásra használt nyelvtannál az lekérdezéseket tartalmazó *eiq* fájl pontos elérési útvonalát kell definiálni. Ez így megfelelő a technika bemutatására, viszont nem használható tényleges produkciós környezetben. Ezért tovább kell gondolni, hogy a jogosultsági rendszert leíró és a lekérdezéseket tartalmazó fájlok mily módon lesznek elérhetőek az éles fejlesztői rendszerek számára és az alapján módosítani a beolvasásukat végző kódrészleteket. Viszont az aktuális kód segítségével látható, hogy a megoldás működőképes és továbbfejlesztésre érdemes.

Ábrák jegyzéke

Irodalomjegyzék

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

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

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

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

[5] Collins-Sussman, Fitzpatrick, Pilato, *Version Control with Subversion, For Subversion 1.7*, Chapter 6. Server Configuration, http://svnbook.red-bean.com/en/1.7/index.html

[6] Chacon, Straub, *Pro Git Second Edition*, Apress (2014), https://git-scm.com/book/en/v2

[7] *MySQL 5.0 Reference Manual*, 6.2 The MySQL Access Privilege System, https://dev.mysql.com/doc/refman/5.0/en/index.html

[8] Oracle, *Database SQL Developer Supplementary Information for MySQL Migrations*, Oracle and MySQL Compared, http://docs.oracle.com/cd/E12151_01/doc.150/e12155/toc.htm

[9] Reddivari, Finin, Joshi, *Policy-Based Access Control for an RDF Store*

[10] Flouris, Fundulaki, Michou, Antoniou, *Controlling Access to RDF Graphs*

[11] AllegroGraph, *AllegroGraph 5.1 Security Implementation*, http://franz.com/agraph/support/documentation/current/security.html

[12] Apache JenaTDB, http://jena.apache.org/index.html

[13] Oracle, *Label Security Administrator's Guide*, http://docs.oracle.com/cd/B28359_01/network.111/b28529/intro.htm

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

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

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

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

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

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

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

[21] EMF-INCQUERY honlapja, https://www.eclipse.org/incquery/

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

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

[24] Bergmann Gábor, Translating, OCL to Graph Patterns Extended Version (2014)

Linkek elérési időpontja: 2015.05.24.

Függelék

[1] A dolgozathoz tartozó forráskód elérési útvonala: https://github.com/FTSRG/mondo-collab-framework/tree/master/mondo-access-control.

[2] 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>
```

[3] policy szabályleírásból generált szabályzat JSON struktúrában:

```
[
  {
    "defaultPermission": "ALLOW",
    "permissionOverride": "DENY",
    "targetType": "GROUP",
    "guid": "designer",
    "policies": [
      {
        "name": "systemInput",
        "permission": "DENY",
        "query": {
          "path":
"d:\\Projects\\MONDO\\hu.bme.mit.ftsrg.mondo.accesscontrol.incquery\\src\\wts
pec\\queries.eiq"
        },
        "pattern": "wtspec.systemInput"
      },
      {
        "name": "systemOutput",
        "permission": "DENY",
        "query": {
          "path":
"d:\\Projects\\MONDO\\hu.bme.mit.ftsrg.mondo.accesscontrol.incquery\\src\\wts
pec\\queries.eiq"
        },
        "pattern": "wtspec.systemOutput"
      },
      {
        "name": "ctrlUnit99",
        "permission": "DENY",
        "query": {
          "path":
"d:\\Projects\\MONDO\\hu.bme.mit.ftsrg.mondo.accesscontrol.incquery\\src\\wts
pec\\queries.eiq"
        },
```

```
        "pattern": "wtspec.ctrlUnit99"
      }
    ]
  },
  {
    "permissionOverride": "ALLOW",
    "targetType": "USER",
    "guid": "alice",
    "policies": [
      {
        "name": "subSystemGenerator_CtrUnit99",
        "permission": "ALLOW",
        "query": {
          "path":
"d:\\Projects\\MONDO\\hu.bme.mit.ftsrg.mondo.accesscontrol.incquery\\src\\wts
pec\\queries.eiq"
        },
        "pattern": "wtspec.subSystemGenerator_CtrUnit99"
      }
    ]
  }
]
```