Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

Dinamikus hozzáférés-vezérlési szolgáltatás modellezése és szimulációja

Készítette: Kovacsics Péter
E-mail: kovacsicspeter@gmail.com

Konzulens: Dr. Pataricza András
Külső konzulens: Bősze Tibor

2008.

Nyilatkozat

Alulírott *Kovacsics Péter*, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Budapest, 2008. május 16.

.....
Kovacsics Péter

Tartalomjegyzék

Kivonat

Napjainkban a vállalatok egyre komplexebbé váló informatikai infrastruktúrája új kihívások elé állítja az informatikai, ezen belül pedig a biztonságért felelős szakembereket is. Egy kellően kiterjedt és heterogén környezetben az informatikai eszközök – legyen szó hardver vagy szoftver eszközről – számbavétele, menedzsmentje és karbantartása legalább akkora kihívást jelent, mint az adatok, szolgáltatások, illetve általában véve bármilyen erőforrás védelme.

Ez utóbbi, kulcsfontosságú feladatot korábban eseti alapon történő szabályozással kezelték. Ez a módszer azonban a rendszeradminisztrátorok számára rengeteg addicionális munkával járt, amely a mai infrastrukturális viszonyok között már kivitelezhetetlenné vált. Ezen kívül a biztonsági szabályok beállítása és betartatása szintén pusztán a rendszeradminisztrátorok felelőssége volt, az ő feladatuk volt a rendszer sértetlenségének garantálása.

Felmerült az igény a biztonsági kérdések házirend alapon történő szabályozására, amellyel nem csupán a nagy méretű infrastruktúra vált jól skálázható módon kezelhetővé, de a biztonsági szabályozások betartatása is kikerült a rendszeradminisztrátorok egyéni felelőssége alól. A házirendek megalkotásában, beállításában és karbantartásában természetesen továbbra is az informatikai és biztonsági szakembereké a felelősség, de azok érvényre juttatásáról már maga az informatikai rendszer gondoskodik.

A hozzáférés szabályozásának házirend alapon történő megvalósítására számos alternatíva terjedt el. Az egyes megoldások közt nem csupán fogalomrendszerükben és használati módjukban, de a szabálytípusok leíró erejében, illetve az implementációk robosztusságában és erőforrásigényében is jelentős különbségek tapasztalhatóak.

Ezen diplomamunka célja áttekintő képet adni az egyes, napjainkban használatos hozzáférés-vezérlési koncepciókról, valamint ismertetni ezen módszerek matematikai modelljét. Bemutatásra kerül egy konkrét implementáció, majd az erre épülő, annak lehetőségeit kibővítő, dinamikus viselkedés leírására képes egyedi megoldás. Ismertetem ennek metamodelljét, szimulálom a jogosultság kiértékelés folyamatát egy adott modellen, majd pedig biztonsági kényszerek, invariánsok teljesülését vizsgálom azon.

Abstract

Nowadays the ever more complex informatics infrastructure of enterprises creates new challenges for the professionals including the security specialists. In an environment extensive and diverse enough, the revision, management and maintenance of informatics tools – both software and hardware – is at least as challenging as the protection of data, services or of any resource in general.

The latter, crucial task had formerly been handled with ad hoc regulations. However, this method had meant tremendous amount of additional workload for the system administrators that has become unattainable among present-day infrastructural circumstances. Besides this the adjustment of and compliance with security regulations had also been the responsibility of the administrators. It was their task to guarantee the integrity of the system.

The necessity occurred to regulate security issues on a policy basis, by which not only the extensive infrastructure has become manageable in a scalable way but the compliance with security regulations has also ceased to be the individual responsibility of the system administrators. Naturally, the informatics and security specialists are still accountable for the creation, set up and maintenance of policies but the enforcement is provided by the informatics system itself.

There are several alternatives of policy based access control implementation available. One may observe significant differences between certain solutions not only in their concept systems or their ways of usage but in their descriptive capacity or in the robustness and resource demand of implementations as well.

The objective of this dissertation is to provide an overview of certain access control concepts in use these days, and to outline the mathematical model of these methods. Besides the discussion of a specific implementation, a unique solution describing dynamic behaviour will also be presented. This applies the implementation in question and further enhances its potential. In this thesis I will outline the meta-model of this particular solution, simulate the procedure of evaluating authorization through a given model, and I will also study the execution of security constraints and invariants using the same model.

1 Bevezető

Napjaink nagyvállalati biztonsági megoldásainak tekintetében két lényeges területről kell szót ejtenünk.

Az első az informatikai rendszer használójának azonosítása és hitelesítése, azaz a *„ki vagy?”* kérdés megválaszolása, amely az autentikáció feladata. Az autentikáció megvalósítására háromféle koncepció terjedt el: a tudás alapú autentikáció, melyhez leggyakrabban valamilyen egyedi azonosító (userid, username, ügyfélszám) és a hozzá tartozó titkos információ (password, passphrase, pinkód) párosának használatára van szükség; birtok alapú autentikáció, amely általában valamilyen egyedi azonosítót hordozó fizikai eszköz (chipkártya, token, kulcs) használatát igényli; valamint a biometrikus azonosítás, amely leggyakrabban hang, ujjlenyomat vagy írisz felismerésen alapul. Ezen módszerek valamelyikével - esetleg ezek kombinációjával - egyértelműen meghatározható a rendszer használójának identitása.

A második feladat ezután annak eldöntése, hogy a (most már azonosított) felhasználó az adott rendszerben milyen műveletek elvégzésére jogosult ill. nem jogosult, azaz a *„mit tehetsz meg?”* kérdés megválaszolása, amely az authorizáció, betartatása pedig a hozzáférés vezérlés feladata. Az authorizációs szabályok megfogalmazása, illetve azok betartatása többféle módon történhet: a szabályok leírhatóak statikus módon, vagy figyelembe vehetnek időben változó paramétereket, hozzáköthetőek egyedi felhasználókhoz illetve erőforrásokhoz, vagy bevezethetőek további absztrakciós szintek a szabályok megfogalmazásában.

Az egyes hozzáférés vezérlési megoldások ennek megfelelően granularitásuk, univerzalitásuk illetve erőforrásigényük szerint igen eltérőek lehetnek. Hogy a konkrét üzleti megoldás során melyiket alkalmazzuk, komoly megfontolás tárgyát kell, hogy képezze.

A következőkben bemutatásra kerülnek az egyes elterjedt hozzáférés vezérlési megoldások, majd ezt követően ezek legújabb irányvonalának, a dinamikus hozzáférés vezérlésnek működése, illetve funkcionális modellje.

1.1 Hozzáférés vezérlési módszerek

1.1.1 Szabad belátás szerinti hozzáférés vezérlés (Discretionary Access Control)

A szabad belátás szerinti hozzáférés vezérlés (Discretionary Access Control, DAC)[1] a hozzáférés vezérlés egyik legelső és egyben sok tekintetben legegyszerűbb megvalósítása: az objektum tulajdonosa által definiált engedélyen alapul, lényege, hogy az egyes védett objektumok tulajdonosa döntheti el, kinek milyen jogosultságai legyenek az objektumra vonatkozóan [3].

Discretionary Access Control-t megvalósító rendszerekben a rendszer valamennyi objektumához egyértelműen hozzárendelhető egy felhasználó, amely az adott objektum tulajdonosa. A legtöbb esetben az objektum tulajdonosa kezdetben annak létrehozója lesz. A tulajdonos teljes jogkörrel rendelkezik az adott objektumra vonatkozóan, illetve az objektumhoz való hozzáférés szabályait ő határozhatja meg. Az objektum vagy erőforrás tulajdonosa egyedi felhasználók számára hozzáférési jogot biztosíthat az erőforrásra vonatkozóan, illetve ezen hozzáférési engedélyeket megvonhatja.

Discretionary Access Control megvalósítható például hozzáférés vezérlési listák (Access Control List, ACL) segítségével (lásd: 1.2.2 fejezet).

1.1.2 Előre meghatározott hozzáférés vezérlés (Mandatory Access Control)

Az előre meghatározott hozzáférés vezérlés (Mandatory Access Control, MAC) rendszer-szinten definiált házirendeken alapuló hozzáférés vezérlési eljárás. A MAC alapú rendszerekben a felhasználók és erőforrások jogosultsági szintekre kerülnek besorolásra, és az egyes érzékenységi szintekhez tartozó erőforrásokhoz csak meghatározott megbízhatósági szintű felhasználók férhetnek hozzá [3].

[1] A Discretionary Access Control fogalmát az Amerikai Védelmi Minisztérium által létrehozott Trusted Computer System Evaluation Criteria (1985) fektette le.

Minden szinthez a jogosultságok egy halmaza rendelhető. Az egyes halmazok közül tetszőleges kettőre igaz az, hogy az egyik szigorú részhalmaza a másiknak, ilyen módon a jogosultsági szintek sorrendezhetőek.

A MAC központi fogalma az érzékenységi címke: minden felhasználóhoz és erőforráshoz (amelyek tipikusan érzékeny adatok) a házirend egy-egy címkét rendel, amely felhasználó esetén a megbízhatóság szintjét, erőforrás esetén pedig annak érzékenységi szintjét határozza meg. Egy adott erőforráson értelmezett művelet elvégzéséhez a kérést indító felhasználónak legalább olyan (vagy magasabb) megbízhatósági szinttel kell rendelkeznie, mint ami az adott szintű erőforráson a kért művelet végrehajtásához szükséges.

Fontos megjegyezni, hogy ugyanazon erőforráson végzett különböző műveletek elvégzéséhez különböző megbízhatósági szint lehet szükséges. Ennek gyakorlati jelentőségét a következő példa szemlélteti: a HR adatok módosítása csak felső szintű jogosultsággal lehetséges, a HR adatok megtekintése közepes szintű jogosultsággal, ugyanakkor a HR adatok alapján egy aggregált nézet megtekintése (pl. hogy hány alkalmazott van összesen) egy alacsonyabb szintű jogosultsággal is engedélyezett lehet.

Annak meghatározása, hogy az egyes érzékenységi szinttel rendelkező erőforrásokon milyen megbízhatósági szinttel rendelkező felhasználók milyen műveleteket végezhetnek, az erőforrások érzékenységi címkéjének legkisebb felső korlátjának és a felhasználók megbízhatósági szintjének legnagyobb alsó korlátjának összerendelésével lehetséges. Ennek leírására a Mandatory Access Control kétféle lehetőséget biztosít:

A szabály-alapú hozzáférés vezérlés (rule-based access control) egyszerű feltételekkel írja le a védett erőforráshoz való hozzáférés szabályait: legfeljebb m érzékenységi szinttel rendelkező erőforráson végzett k művelethez legalább n megbízhatósági szinttel kell rendelkeznie a felhasználónak. A hozzáférés vezérlés ezen formáját valamennyi MAC-alapú hozzáférés vezérlési rendszer megvalósítja; ilyenkor pusztán egy felhasználói és egy védett erőforrás érzékenységi címkéjének összehasonlításával dől el az adott művelet engedélyezése vagy visszautasítása.

Mátrix-alapú hozzáférés vezérlés (lattice-based access control) segítségével pedig összetett, több erőforrást és felhasználót is érintő döntési szabályokat fogalmazhatunk meg: a hozzáférési mátrix soraiban a hozzáférést kérő szereplők

megbízhatósági címkéi, oszlopaiban a hozzáférhető erőforrások érzékenységi címkéi találhatóak, a mátrix elemei pedig az adott szereplő számára az adott erőforráson engedélyezett műveletek szerepelnek.

Kritikus kérdés ugyanakkor az adatok importálása és exportálása: más rendszerekből való adat-importálás illetve más rendszerekbe való exportálás (ide tartozik még a nyomtatás is!) kritikus funkció a MAC-alapú rendszerekben, mivel ezen folyamatok során különös tekintettel kell lenni az érzékenységi címkék helyes kezelésére, hogy az érzékeny adatok védelme továbbra is biztosított legyen.

1.1.3 Szerepkör-alapú hozzáférés vezérlés (Role-based Access Control)

A szerepkör-alapú hozzáférés vezérlés (Role-based Access Control, RBAC) rendszer-szintű házirendeken alapuló hozzáférés vezérlési eljárás. A hozzáférés szabályozása szerepkörök alapján történik: az egyes felhasználók egy vagy több szerepkört tölthetnek be, a szerepkörökhöz pedig jogosultságok egy-egy készletét rendeljük, ilyen módon az egyes felhasználók közvetett módon, a szerepkörükön keresztül nyerhetik el a különböző jogosultságokat [4][5].

Minden felhasználóhoz szerepkörök egy halmaza van rendelve. A felhasználó az egyes munkamenetek során ebből a halmazból aktiválhat szerepköröket, mellyel a munkamenet elnyeri az aktivált szerepkörökhoz tartozó jogosultságokat.

A szerep-alapú hozzáférés vezérlés működése ennek megfelelően három egyszerű szabállyal leírható:

- Minden felhasználóhoz hozzárendelünk legalább egy szerepkört.
- Minden felhasználó csak a hozzárendelt szerepkörök valamelyiké(i)t aktiválhatja.
- Minden felhasználó csak azon műveleteket hajthatja végre, amelyeket az aktív szerepe(i) lehetővé tesz(nek).

Mivel a jogosultságokat nem közvetlenül az egyes emberekhez, hanem szerepekhez kötjük, a hozzáférések kezelése nagyban egyszerűsödik. Például új munkatárs felvétele vagy éppen szervezeti egység váltása esetén elegendő a felhasználó szerepköreit megváltoztatni, és jogosultságai ezzel együtt módosulni fognak.

A szerepkör-alapú hozzáférés vezérlés a Mandatory Access Control-hoz hasonlóan több felelősségi kört kezelő rendszerek esetén használatos, de a jogosultságok kezelésének tekintetében jóval szélesebb lehetőségeket kínál annál.

Mivel számos vállalati és törvényi szabályozás is megköveteli a jogosultságok betartatásának, ezen belül is a felelősségek szétválasztásának informatikai eszközökkel való támogatását, ezért minden vállalatnak szüksége van ezt megvalósító megoldásokra. Emiatt a legtöbb szoftvergyártó rendelkezik az ezen a téren legrugalmasabb, szerepkör-alapú hozzáférés vezérlést megvalósító termékkel, mint például az IBM Tivoli Access Manager, CA eTrust Admin, Novell Web Access Manager.

1.1.4 Egyéb megközelítések, kiterjesztések

1.1.4.1 Hierarchikus, szerepkör-alapú hozzáférés vezérlés (Hierarchical Role-based Access Control)

A hierarchikus szerepkör-alapú hozzáférés vezérlés abban különbözik a hagyományostól, hogy lehetővé teszi a szerepkörök hierarchiába szervezését, egymással való kombinálását. Ilyenkor egy szerepkörhöz nem csak felhasználók, hanem más szerepkörök is rendelhetőek, a szerepkörökhöz rendelt jogosultságok pedig a szerepkörök hierarchiájának megfelelően öröklődhetnek: ha egy felhasználót egy szerepkörhöz rendelünk, akkor ő nem csak az ahhoz, hanem az „őseihez” kapcsolt jogosultságokat is elnyeri. Ez által a hozzáférési szabályok tovább finomíthatóak, rugalmasabban menedzselhetőek.

1.1.4.2 Dinamikus, szerepkör-alapú hozzáférés vezérlés (Dynamic Role-based Access Control)

A dinamikus, szerepkör-alapú hozzáférés vezérlés a hagyományos szerepkör-alapú hozzáférés vezérlés megoldását kiegészíti azzal a képességgel, hogy az authorizációs szabályok kiértékelésének folyamatába (a kérést indító személyén és a kérés célján kívül) a művelet végrehajtásának környezetéből származó paramétereket is bevonjunk. Ezen paraméterek tetszőleges, a döntés kontextusából származó információk - mint például a kérés időpontja -, vagy környezeti változók lehetnek, melyek a döntési szabályok megfogalmazásában új megközelítést jelentenek.

Segítségükkel sokkal finomabb authorizációs szabályok írhatóak le, úgymint hogy *ki, milyen műveletet végezhet, mely védett erőforráson, valamint milyen környezeti paraméterek mellett*. Az ilyen módon, környezeti paraméterekkel kibővített jogosultság kiértékelési folyamat modell szinten is leképezhetővé válik, amelyen ezután (a környezeti paramétereket is figyelembe vevő) formális ellenőrzések végezhetőek.

A környezeti paraméterek figyelembe vételének képességével felruházott szerepkör-alapú hozzáférés vezérlést megvalósító szolgáltatások a szabályok megfogalmazásában nagy mértékű rugalmasságot biztosítanak. Az ilyen irányú kezdeményezések nem újkeletűek az ipari szoftver-megoldásokban, de ezek nem kellően egységesek és formalizálhatóak. A dinamikus, szerepkör-alapú hozzáférés vezérlés képes az egyedi kiterjesztések által nyújtott szabályrendszerek formalizálására és modell szinten történő leírására, amely napjainkban még merően új lehetőséget jelent [6].

1.1.4.3 Környezetfüggő hozzáférés vezérlés (Context-based Access Control)

A kontextusfüggő hozzáférés vezérlés[2] egy további megközelítést jelent a hozzáférés vezérlési módszerek között. Segítségével a jogosultsági döntés olyan TCP és UDP csomagokból kinyerhető alkalmazás-rétegbeli információk alapján is megtehető, minthogy az adott kérés intraneten vagy az Interneten keresztül, illetve hogy milyen IP-címtartományból érkezett. Ezen hozzáférés-vezérlési megközelítés segítségével tehát a kérést igénylő személyén kívül már egyéb, a kérés kontextusából származó információk is bevonhatóak a jogosultság-ellenőrzési folyamatba.

Az ezen funkcionalitást megvalósító intelligens szűrők tűzfalba építhető módon konfigurálhatóak, egyszerű szabályok definiálásával [7].

[2] A fogalmat a Cisco vezette be a hálózati forgalom szabályozása területén.

1.1.4.4 Nézet-alapú hozzáférés vezérlés (View-based Access Control)

A nézet-alapú hozzáférés vezérlés (View-based Access Control, VACM)[3] az SNMP v3[4] változatában megjelent, a hálózati rétegben értelmezett hozzáférés-vezérlési megoldás. Lényege a *Management Information Base* (MIB) objektumokhoz való hozzáférés a felhasználók ún. MIB nézetekhez (*MIB view*) való rendelésével történő szabályozása [8].

A Context-based Access Control és a View-based Access Control módszerek gyakorlati alkalmazás szempontjából nagy jelentőségűek, ugyanakkor formális modellel nem rendelkeznek.

[3] A fogalmat B. Wijnen (IBM), R. Presuhn (BMC) és K. McCloghrie (Cisco) alkotta meg az 1999-ben kiadott RFC2575-ben.
[4] Simple Network Management Protocol v3, B. Wijnen (IBM), R. Presuhn (BMC) és K. McCloghrie (Cisco), RFC2571

1.1.5 Hozzáférés vezérlési módszerek összehasonlítása

	Discretionary Access Control	Mandatory Access Control	Role-based Access Control
A hozzáférés vezérlés tárgya	Erőforrás	Erőforrás	Erőforrás
A hozzáférés vezérlésének alapja	Explicit engedély	Megbízhatósági címke	Betöltött szerepkör
A megvalósított jogosultsági halmazok	-	Lineáris jogosultsági szintek	Egymást átfedhető jogkörök
A hozzáférés meghatározója	Tulajdonos / létrehozó	Rendszer / Adminisztrátor	Rendszer / Adminisztrátor vagy más felhasználó*

* Számos implementációban a jogosultságok beállítása szintén olyan művelet, amely engedélyezhető szerepkörök számára. Ilyen módon a delegált adminisztráció valósítható meg, sőt szélsőséges esetben akár DAC is.

1.2 A hozzáférési jogosultságok ábrázolása: elterjedt adatszerkezetek

Az előző fejezet átfogó képet nyújtott az egyes hozzáférés-vezérlési módszerekről. Ezen módszerek lefektetik a hozzáférés szabályozásának irányelveit, de a jogosultságok ábrázolásának részleteire nem térnek ki.

A hozzáférési jogosultságok ábrázolására többféle gyakorlat terjedt el. Ezek közül a legkorábbi a UNIX fájlrendszereken használt jogosultsági struktúra, amely rögzíti a fájl tulajdonosának, csoportjának és bárki másnak olvasási, írási és végrehajtási jogosultságait. Ennek korlátozásait áthidalandó alakult ki a hozzáférés vezérlési listák (Access Control List, ACL) adatszerkezete, amely lehetővé teszi tetszőleges számú felhasználó és csoport, valamint művelet kezelését.

Az ACL-ek kiterjeszthetőek olyan módon, hogy az egyes műveletekhez őrfeltételként logikai kifejezést rendeljünk, így a feltétel teljesülése határozza meg, hogy a művelet végrehajtható-e.

1.2.1 A hagyományos UNIX fájlrendszer jogosultságok

A hozzáférési szabályok ábrázolására az első, igazán széles körben elterjedt módszer a hagyományos UNIX rendszerek „rwx bit”-jeinek alkalmazása. A UNIX alapú rendszerekben az egyes fájlokhoz és könyvtárakhoz egy 3x3 (+3 bit)[5] bitből álló mező kapcsolódik, amely meghatározza, hogy a fájlon vagy könyvtáron végezhető mely műveletekre ki jogosult [12].

A műveletek tekintetében megkülönböztetünk írást, olvasást, végrehajtást és a tulajdonos (csoport) jogaival való futtatást, a felhasználók tekintetében pedig a fájl/könyvtár tulajdonosát, a fájlhoz rendelt felhasználói csoport tagjait, és az ebbe a két kategóriába nem tartozó többi felhasználót.

Az egyes állományokhoz kapcsolt három, hárombites bitmező rendre a tulajdonos, a fájl csoportjának tagjainak, illetve a más felhasználók hozzáférési szabályait írják le: a három bit közül az első az olvasás, a második az írás, a harmadik fájloknál a végrehajtás, könyvtáraknál a listázás jogosultságát reprezentálja: azon

[5] suid, guid, sticky bit

műveletek engedélyezettek a felhasználók számára, amelyekhez tartozó bit értéke ponált.

Példa: a /etc/passwd fájlhoz kapcsolt

```
rwx rw- r--
```

bitsorozat azt jelenti, hogy tulajdonosa rendelkezik rajta olvasási, írási és végrehajtási joggal is (`rwx`), a fájlhoz rendelt felhasználói csoport tagjai rendelkeznek rajta írási és olvasási joggal (`rw-`), a többi felhasználónak pedig csak olvasási joga van a fájlra vonatkozóan (`r--`).

Az rwx jogosultsági bitek esetében a hármas bitsorozatokat számokkal szokás helyettesíteni, ilyen módon az 3x3 bit leírható 3x1 számjeggyel, az előbbi példánál maradva:

```
7 6 4
```

A UNIX rendszerekben a jogosultsági bitek beállítását az egyes állományok tulajdonosai, vagy a root felhasználó végezhetik, emiatt az ilyen rendszerek a Discretionary Access Control és a Mandatory Access Control lehetőségeit egyesítő, Role-based Access Control kategóriájába sorolhatóak.

Mivel a UNIX jogosultsági bitjei a felhasználóknak csak három kategóriáját (tulajdonos, tulajdonos csoportjának tagjai és mások) és háromféle műveletet (olvasás, írás, végrehajtás) különböztetnek meg, a hozzáférési szabályok finomításával együtt felmerült az igény egy, az „rwx” biteknél rugalmasabb leírási mód kidolgozására. Ezen igény kielégítésére alakult ki a hozzáférés vezérlési listák használata.

1.2.2 Hozzáférés vezérlési lista (Access Control List)

Az ACL (Access Control List – Hozzáférés vezérlési lista) alapú jogosultság-betartatás egy széleskörben elterjedt technikai megvalósítása a DAC, MAC és RBAC alapú jogosultsági modellek által leírt hozzáférési korlátozások kikényszerítésére. Az ACL egy olyan hozzáférési szabályokat leíró struktúra, amelyet a védett erőforráshoz meta-adatként kapcsolva leírja az adott erőforrás hozzáférési házirendjét. Egy ACL olyan ACL-bejegyzések listája, amelyek azt rögzítik, hogy milyen alany (felhasználó, felhasználói csoport) milyen műveletet (hozzáférési módot, operációt) hajthat végre azon a védett objektumon, amelyre az ACL vonatkozik: például az (Jakab, olvasás)

bejegyzés a jelszavak.txt fájl ACL-jében azt jelenti, hogy Jakab olvashatja a jelszavak.txt fájlt.

Egyes implementációk megengedik tiltó és engedélyező ACL-bejegyzések megadását míg más implementációk az ACL-ek öröklését[6] teszik lehetővé. Ezek a kiterjesztések kényelmes, szemantikusabb házirend-kialakítást tesznek lehetővé illetve elősegítik a házirend kompaktabb tárolását – ábrázolását, de a kiterjesztett ACL-alapú házirendek leíró ereje minden esetben megegyezik a hagyományos ACL-ekével.

Egy ACL-alapú biztonsági rendszerben egy erőforráshoz való hozzáféréskor maga a rendszer ellenőrzi az erőforráshoz kapcsolt valamennyi ACL-bejegyzést, és eldönti, a kérés indítója jogosult-e az igényelt műveletre az erőforrásra vonatkozóan.

Az igényelt műveletek lehetnek egyeleműek (pl. olvasás) vagy összetettek (pl. olvasás-írás), a döntés azonban mindkét esetben kétértékű lehet: engedélyezés vagy elutasítás. Összetett művelet csak akkor kerül engedélyezésre, ha annak valamennyi része engedélyezett.

Például: ha a (Jakab, olvasás, jelszavak.txt) kérés eredménye (engedélyezés), a (Jakab, írás, jelszavak.txt) kérésé (elutasítás), akkor a (Jakab, olvasás-írás, jelszavak.txt) kérés eredménye is (elutasítás) lesz.

Csupán engedélyező típusú ACL-eket használó implementáció esetén az egyes kérések jogosultság kiértékelése során figyelembe veendő ACL bejegyzések halmaza a vonatkozó engedélyező ACL bejegyzések uniója.

- Ez egyelemű művelet kérése esetén azt jelenti, hogy a kiértékelést elegendő az első, a hozzáférést megengedő ACL bejegyzésig végezni, hiszen az első engedélyező bejegyzés után a döntés mindenképpen pozitív lesz, és a művelet végrehajtásra kerül. Ha az ACL bejegyzések között egyetlen, a kérést megengedő sem található, a kérés elutasításra kerül.

- Összetett művelet igénylése esetén az engedélyezett műveletek halmaza az lesz, amelyekre vonatkozóan az ACL tartalmaz engedélyező bejegyzést, ehhez a kiértékelést

[6] Például szülő-mappa fájlrendszer esetében

addig kell folytatni, amíg az összes műveletre vonatkozóan nem találunk engedélyező ACL bejegyzést, vagy az ACL bejegyzések végére nem érünk.[7]

Tiltó típusú ACL-eket is használó implementáció esetén az egyes kérések jogosultság kiértékelése során figyelembe veendő ACL bejegyzések halmaza a vonatkozó engedélyező ACL bejegyzések uniója mínusz a vonatkozó tiltó ACL bejegyzések halmaza (a tiltó ACL bejegyzések erősebbek), vagyis azok lesznek az engedélyezett műveletek, amelyekre vonatkozóan található engedélyező bejegyzés az ACL-ek között, de nem található tiltó bejegyzés.

- Ez egyelemű művelet esetén azt jelenti, hogy a kiértékelés az első, a hozzáférést tiltó ACL bejegyzésnél leállhat, ilyenkor a döntés negatív lesz, és a kérés elutasításra kerül. Ha az ACL bejegyzéseken végigiterálva, a bejegyzések között egyetlen, a kérést megtiltó sem található, viszont található közöttük legalább egy, amely a kérést engedélyezi, úgy a művelet végrehajtásra kerül az erőforráson.
- Összetett művelet igénylése esetén az engedélyezett műveletek halmaza az lesz (valamennyi ACL-bejegyzés figyelembe vételével), amelyekre vonatkozóan az ACL tartalmaz engedélyező bejegyzést, de nem tartalmaznak tiltó bejegyzést.

Egy ilyen rendszerben természetesen kulcskérdés az ACL-ek meghatározása: az egyes objektumokra vonatkozóan ki és milyen módon szerkesztheti az ACL-ek tartalmát. Az ACL-alapú rendszerek ezalapján mind a Discretionary, mind a Mandatory, mind pedig a Role-based Access Control kategóriájába besorolhatóak.

ACL-alapú rendszerek esetén Discretionary Access Control-ról akkor beszélhetünk, ha a létrehozó vagy tulajdonos teljes jogkörrel rendelkezik az objektumra vonatkozóan, beleértve a hozzá kapcsolt ACL-ek, vagyis a hozzáférési szabályok módosítását is. Mandatory Access Control-ról beszélünk akkor, ha rendszer-szintű szabályok vagy rendszeradminisztrátorok határozzák meg ACL-eket az egyes erőforrásokra vonatkozóan, nem pedig azok tulajdonosa. Ugyanakkor az ACL-alapú rendszerek Role-based Access Control-t valósítanak meg abban az esetben, ha az ACL-ek, és ezzel együtt a jogosultságok beállítása a rendszeradminisztrátorok felelőssége, amelyet azonban delegálhatnak az egyes felhasználóknak, így ők is szerkeszthetik az ACL-ek tartalmát.

[7] Egy felhasználó több csoportnak is tagja lehet, előfordulhat, hogy míg az egyik csoporttagsága az ACL bejegyzések alapján megengedi számára az igényelt műveletek egy részét, egy másik csoporttagsága pedig azok más részét.

ACL-alapú hozzáférés vezérlést tartalmaz a Microsoft Active Directory, az IBM Tivoli Directory Server és számos más címtár-megoldás; ezenkívül a Linux operációs rendszerek *ext2*-es vagy újabb fájlrendszer használata esetén lehetővé teszik az ACL-ek használatát a fájlok hozzáférés-vezérlésének szabályozására.

1.2.3 Egyéb ábrázolási módok

A hozzáférési szabályok leírására az ACL-ek alkalmazása az egyik legelterjedtebb módszer. A hagyományos ACL-ek statikusan rögzítik, hogy ki – milyen művelet elvégzésére jogosult. Az ACL-ek kiterjeszthetőek olyan módon, hogy a hagyományos ACL bejegyzések jogosultságvektora helyett minden művelethez egy-egy őrfeltételt – logikai kifejezést - rendeljünk, amely kiértékelésének eredménye határozza meg, hogy az adott műveletet engedélyezi-e a bejegyzés.

Ezen őrfeltételek olyan bemeneti paramétereket használnak, amelyek futásidőben helyettesítődnek be, így a feltétel kiértékelése nagyobb leíróerőt tesz lehetővé, mivel figyelembe veheti a kiértékelés kontextusát.

2 Modellezés

Az egyes hozzáférés-vezérlési megoldások történetileg egymásra épülve alakultak ki: az idő teltével exponenciálisan emelkedett a jogosultság-kezelés és hozzáférés-vezérlés hatókörébe tartozó entitások (például az objektumok, engedélyek és szerepkörök) száma, ez egyre komplexebb biztonsági házirendeket eredményezett.

Az újabb hozzáférés vezérlési módszerek, az egyes megvalósítások egyre nagyobb hangsúlyt fektettek a karbantarthatóságra, a változások könnyű kezelésére – például szerepkörök, csoportok vagy sablonok segítségével, de a komplexitás robbanásszerű növekedése miatt a vállalati hozzáférési házirendek elérték azt a mértéket, amely a hagyományos, manuális módon már nem karbantartható úgy, hogy az egyidőben konzisztens és költséghatékony is lehessen. Igény mutatkozott ugyanakkor a házirendek modell alapú, formális validálhatóságára, verifikálhatóságára.

További problémát jelent, hogy az egyes gyártók megoldásaikban olyan kiterjesztéseket alkalmaznak, amelyek a modell által le nem írt, futás idejű paramétereket vezetnek be a kiértékelési folyamatokban. Mivel ezek az „extra” szabályok nem jelennek meg modell szinten, nem beszélhetünk modell alapú validálhatóságról, hiszen a modell nem a tényleges működést tükrözi.

Számos kutatás irányul az informatikai rendszerek jogosultság-kezelésének modellezésére, annak validálására – erre irányulnak a Bell - La Padula, Chinese Wall, Clark-Wilson biztonsági modellek – de sok esetben kimarad annak a tárgyalása, hogy az egyes gyártók egyedi kiegészítései, megoldásai miként vezethetők vissza, miként írhatók le a fent írt módszerekkel. Ez a visszavezetés tenné lehetővé azt, hogy visszavezessük a modellbe a műszakilag indokolt „extra” kiegészítéseket (pl. az objektum térbe történő kihajtogatás segítségével).

Ezért tekintjük most át a legelterjedtebb biztonsági modelleket, majd a korábban megismertetett hozzáférési módszerek jogosultsági modelljeit.

2.1 Biztonsági modellek

2.1.1 Bell - La Padula

A Bell – La Padula biztonsági modell[8] szorosan összekapcsolódik a Mandatory Access Control fogalmával: annak meghatározása, hogy az egyes felhasználók mely erőforrásokon mely műveleteket végezhetnek, a felhasználók és erőforrások biztonsági szintjének összehasonlítása alapján történik, az egyes felhasználók ill. objektumok biztonsági szintjét pedig az azokhoz kapcsolt címkék határozzák meg [10].

A modell a hozzáférés szabályozására két alapvető szabályt definiál:

- „No-read-up”: az egyes felhasználók nem olvashatják a biztonsági szintjüknél magasabb besorolású adatokat.
- „No-write-down”: az egyes felhasználók nem írhatják a biztonsági szintjüknél alacsonyabb besorolású adatokat.

Ezen szabályok együttese azt jelenti, hogy az egyes felhasználók adatokat csak a saját, vagy magasabb biztonsági szintjükön hozhatnak létre; illetve adatokat csak a saját, vagy alacsonyabb biztonsági szintjükről olvashatnak, ezzel megakadályozható a bizalmas információk magasabb szintről alacsonyabb szintre történő áramlása.

2.1.2 Biba

A Biba modell[9] a Bell – La Padula modellhez hasonlóan a Mandatory Access Control fogalmaiból építkezik: a felhasználók és erőforrások biztonsági szintjének (amelyet a hozzájuk kapcsolt címke határoz meg) összehasonlítása alapján történik a jogosultság kiértékelése. Ám amíg a Bell – La Padula modellben a lefelé írás, illetve a felfelé olvasás tiltott műveletek voltak, addig a Biba modell ebben éppen duálisa, ezek az általa definiált engedélyezett műveletek.

A modell működését meghatározó szabályok:

- „No-read-down”: az egyes felhasználók nem olvashatják a biztonsági szintjüknél alacsonyabb besorolású adatokat.

[8] David Elliott Bell és Len La Padula, 1973
[9] Kenneth J. Biba,1977

- „No-write-up”: az egyes felhasználók nem írhatják a biztonsági szintjüknél magasabb besorolású adatokat.

Ezen szabályok együttese – a Bell – La Padula modellel ellentétben - azt jelenti, hogy az egyes felhasználók adatokat csak a saját, vagy alacsonyabb biztonsági szintjükön hozhatnak létre; illetve adatokat csak a saját, vagy magasabb biztonsági szintjükről olvashatnak.

2.1.3 Chinese Wall

A Kínai fal modell[10] biztosítja az egymással érdekellentétben[11] lévő felhasználók adatainak elválasztását. Ez a szétválasztás ihlette a modell nevét [11].

A modell szerint az adathozzáférést olyan módon kell irányítani, amely nem teszi lehetővé, hogy ellentétes érdekeltségű felek adatai egy kézbe kerüljenek, például egy banknál az egymással konkurens vállalatok számláját nem kezelheti azonos ügyintéző.

Ennek érdekében a rendszer résztvevői (legyenek azok felhasználók vagy vállalatok) CoI kategóriákba kerülnek besorolásba: egy résztvevő csak egy kategóriának lehet tagja, egy kategóriába pedig kettő vagy több résztvevő tartozhat. Ezekután a hozzáférés jogosultságát három egyszerű szabály írja le:

- Ha a hozzáférés igénylője korábban nem végzett olvasási műveletet, bármely (a számára egyéb szabályok szerint engedélyezett) felhasználó adatait olvashatja.
- Ha a hozzáférés igénylője már végzett olvasási műveletet egy felhasználó adatain, azon felhasználó CoI kategóriájába tartozó többi felhasználó adatait a későbbiekben már nem olvashatja.
- Minden, a szabályrendszer védelmébe nem tartozó információ publikus, azaz szabadon olvasható.

Amint látható, a Kínai fal modell csak az olvasási műveleteket szabályozza, írási és olvasási műveletek kezelésére a Brewer-Nash modell került kidolgozásra [3].

[10] D. Brewer és M. Nash, 1989
[11] Conflict of Interest, CoI

2.1.4 Clark-Wilson

A Clark-Wilson modell[12] lényege, hogy a védett adatokhoz csak ellenőrzött eljárásokon keresztül lehet hozzáférni, az egyes ellenőrzött eljárásokat pedig csak az arra jogosultak hívhatják meg, így garantálva az adatok integritását.

Ehhez a modell az adatokat két kategóriába osztja: védett adatokra (Constrainted Data Item, CDI) és nem védett adatokra (Unconstrainted Data Item, UDI). A védett adatok több osztályba sorolt érzékeny adatok, a nem védett adatokat pedig tipikusan a felhasználói bemenetek jelentik.

Nem védett adat vagy védett adatok egy osztályából származó adat csak megbízható eljárás (Transformation Procedure, TP) hatására kerülhet át védett adatok egy másik osztályába. A megbízható eljárások garantálják a rendszer integritását, azaz a rendszert érvényes állapotból érvényes állapotba kell hogy juttassák. A rendszer integritásának ellenőrzését speciális eljárások (Integrity Verification Procedure, IVP) végzik.

2.2 Hozzáférés-vezérlési szolgáltatások modellezése

Jelen fejezetben bemutatásra kerül az egyes, fent említett hozzáférés-vezérlési modellek formalizálása, valamint ismertetésre kerülnek azok legfőbb tulajdonságai.

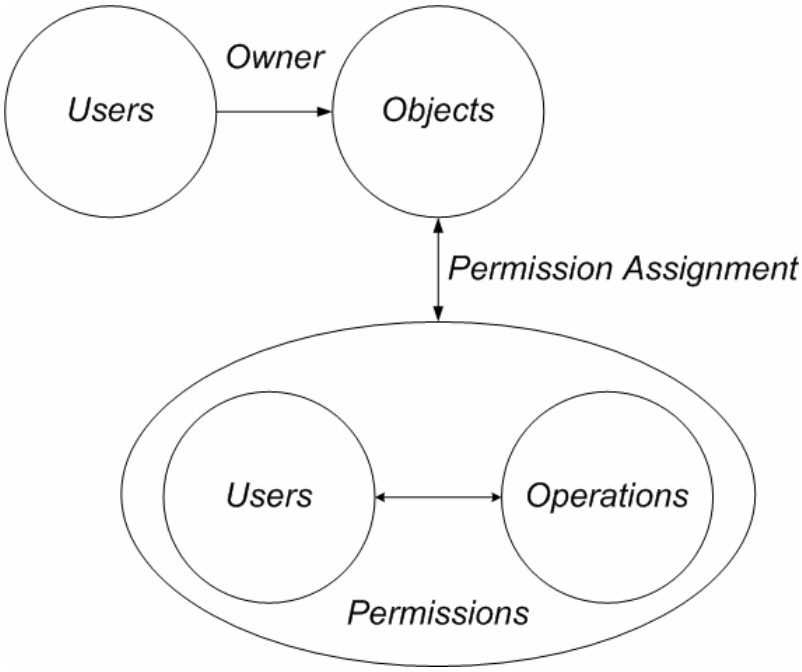
2.2.1 Szabad belátás szerinti hozzáférés vezérlés (Discretionary Access Control)

A Discretionary Access Control modelljének három fő entitása: a felhasználók, a műveletek és az objektumok. Minden objektumhoz egy felhasználót rendelünk, amely felhasználó az objektum tulajdonosa. Ő szerkesztheti az objektumhoz tartozó hozzáférési szabályokat. A hozzáférési szabályok felhasználó-művelet párosok listájából állnak.

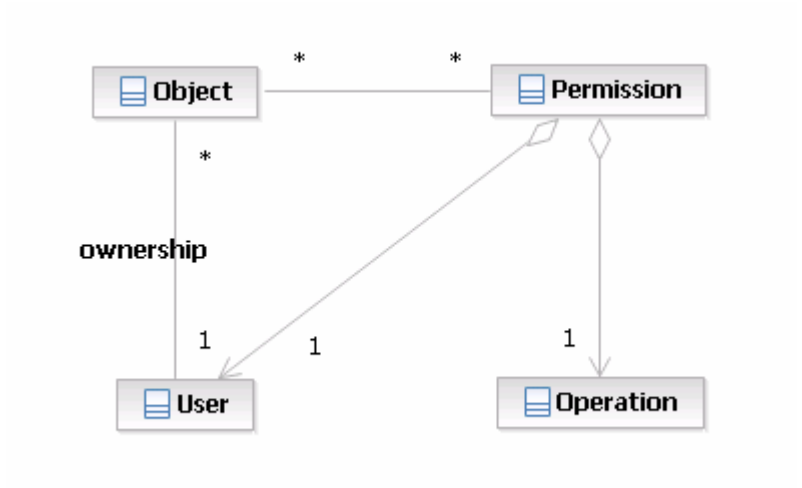
A felhasználó *B* műveletet *C* erőforráson akkor hajthat végre, ha az (*A*, *B*) páros szerepel a *C* erőforrás hozzáférési szabályai között, vagyis engedélyezett kérés az,

[12] David D. Clark és David R. Wilson, 1987

amelyhez tartozó felhasználó-művelet bejegyzés szerepel a kérés célját reprezentáló objektum hozzáférési szabályai között.



1. ábra: A Discretionary Access Control metamodellje



2. ábra: A Discretionary Access Control UML metamodellje

Formálisan:

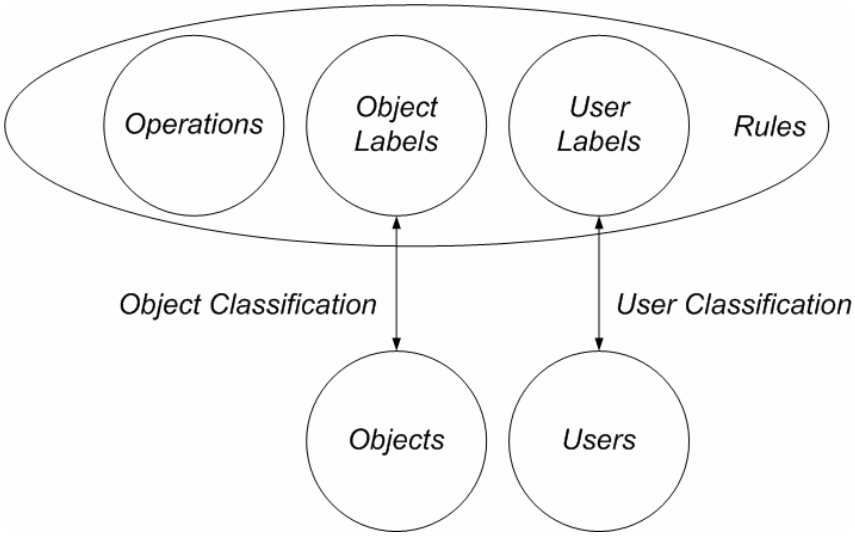
1. USERS, OPS, OBS: felhasználók, műveletek és objektumok.
2. Tulajdonlás:
   $OO \subseteq USERS \times OBS$, $\forall (obj \in OBS)$: { #(u, obj) = 1 | (u, obj) $\in$ OO, u $\in$ USERS}, felhasználók és objektumok összerendelése (egy-a-többhöz kapcsolat, egy objektumnak egy tulajdonosa van).
3. Egy objektum tulajdonosa:
   (obj:OBS) $\rightarrow$ (u:USERS), owner(obj) = { u $\in$ USERS | (u, obj) $\in$ OO }
4. $PRMS = 2^{(USERS \times OPS)}$, a jogosultságok készlete.
5. $PA \subseteq PRMS \times OBS$, jogosultságok és objektumok összerendelése (több-a-többhöz kapcsolat).
6. Egy objektumhoz rendelt jogosultságok:
   (obj:OBS) $\rightarrow$ 2^{PRMS}, assigned_permissions(obj) = { p $\in$ PRMS | (p, obj) $\in$ PA}.
7. Egy felhasználó számára engedélyezett hozzáférések:
   (u:USERS) $\rightarrow$ (op:OPS,obj:OBS), permissions(u) = { op $\in$ OPS, obj $\in$ OBS | (u, op) $\in$ assigned_permissions(obj) }.
8. Hozzáférés jogosultságának ellenőrzése:
   (u:USERS, obj:OBS, op:OPS) $\rightarrow$ (TRUE | FALSE),
   check_permission(u, obj, op) = (
   (u$\in$ USERS) ^
   (obj $\in$ OBS) ^
   (op $\in$ OPS) ^
   ($\exists$ (p = (u, op)) $\in$ PRMS | (p, obj) $\in$ PA).

2.2.2 Előre meghatározott hozzáférés vezérlés (Mandatory Access Control)

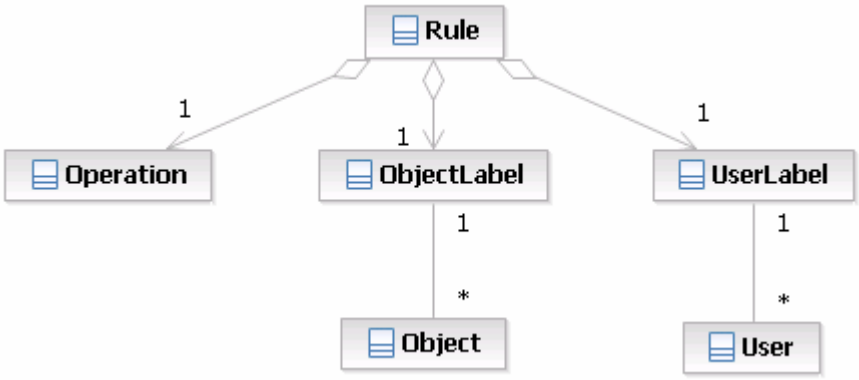
A Mandatory Access Control modelljének fő entitásai: a felhasználók, műveletek, objektumok, valamint az érzékenységi címkék. Az egyes felhasználókhoz egy érzékenységi címkét rendelünk, amely a felhasználó megbízhatósági szintjét jelöli. Az egyes erőforrásokhoz is egy érzékenységi címkét kapcsolunk, amely az erőforrás érzékenységi szintjét jelöli.

A hozzáférési szabályokat rendszer-szinten definiáljuk. A hozzáférési szabályok műveletekből, felhasználói és erőforrás érzékenységi címkékből álló hármasok, amelyek meghatározzák, az adott érzékenységi szintű erőforráson az adott művelet elvégzésére milyen megbízhatósági szintű felhasználók jogosultak.

A felhasználó (akinek megbízhatósági szintje *n*) *B* műveletet *C* erőforráson (amelynek érzékenységi szintje *m*) akkor hajthat végre, ha a (*B, n1, m1*) hármas szerepel a rendszer szintű hozzáférési szabályok között, ahol $n \geq n1$ és $m \leq m1$, vagyis engedélyezett kérés az, amelynek felhasználói megbízhatósági szintjénél nem kisebb és erőforrás érzékenységi szintjénél nem nagyobb bejegyzés szerepel a hozzáférési szabályok között.



3. ábra: A Mandatory Access Control metamodellje



4. ábra: A Mandatory Access Control UML metamodellje

Formálisan:

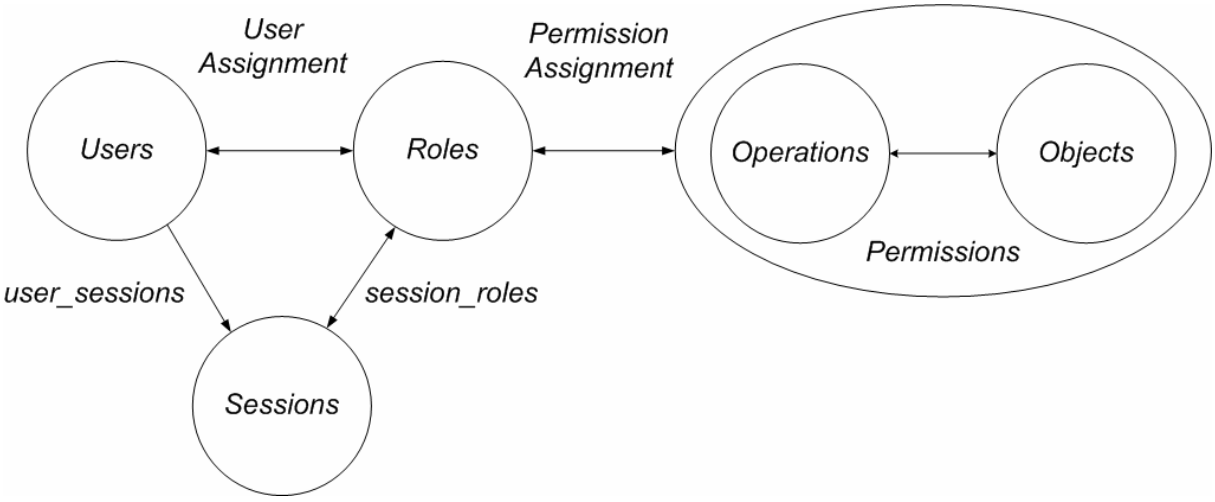
1. USERS, ULABELS, OLABELS, OPS, OBS: felhasználók, felhasználói és erőforrás érzékenységi címkék, műveletek és objektumok.
2. Felhasználó minősítése:
   UA $\subseteq$ USERS $\times$ ULABELS, felhasználók és érzékenységi címkék összerendelése (egy-a-többhöz kapcsolat).
3. Egy felhasználó megbízhatósági szintje:
   (u:USERS) $\rightarrow$ (s:ULABELS), label(u) = { s $\in$ ULABELS | (u, s) $\in$ UA }.
4. Erőforrás minősítése:
   OA $\subseteq$ OBS $\times$ OLABELS, objektumok és érzékenységi címkék összerendelése (egy-a-többhöz kapcsolat).
5. Egy erőforrás érzékenységi szintje:
   (obj:OBS) $\rightarrow$ (s:OLABELS), label(obj) = { s $\in$ OLABELS | (obj, s) $\in$ OA }.
6. Hozzáférési szabályok:
   PY $\subseteq$ OPS $\times$ ULABELS $\times$ OLABELS, műveletek, felhasználói és erőforrás érzékenységi címkék összerendelése (több-a-több-a-többhöz kapcsolat).
7. Egy felhasználó számára engedélyezett hozzáférések:
   (u:USERS) $\rightarrow$ (op:OPS,obj:OBS), permissions(u) = { op $\in$ OPS, obj $\in$ OBS | (op, us1, os1) $\in$ PY | us = labels(u), os = labels(o), us $\geq$ us1, os $\leq$ os1 }.
8. Hozzáférés jogosultságának ellenőrzése:
   (u:USERS, obj:OBS, op:OPS) $\rightarrow$ (TRUE | FALSE),
   check_permission(u, obj, op) = (
   (u $\in$ USERS) ^
   (obj $\in$ OBS) ^
   (op $\in$ OPS) ^
   (us $\in$ ULABELS | us = label(u)) ^
   (os $\in$ OLABELS | os = label(obj)) ^
   ($\exists$(obj, us, os)$\in$ PY).

A Mandatory Access Control jogosultsági modellje megvalósítja a Bell – La Padula és a Biba biztonsági modellek által támasztott követelményeket: az általa definiált jogosultsági szintek és az azokhoz kapcsolódó hozzáférési szabályok a felelősségek szétválasztását, illetve a Bell – La Padula modellben megfogalmazott lefelé olvasás és felfelé írás, valamint a Biba modellben meghatározott lefelé írás és felfelé olvasás megakadályozását is megvalósítják.

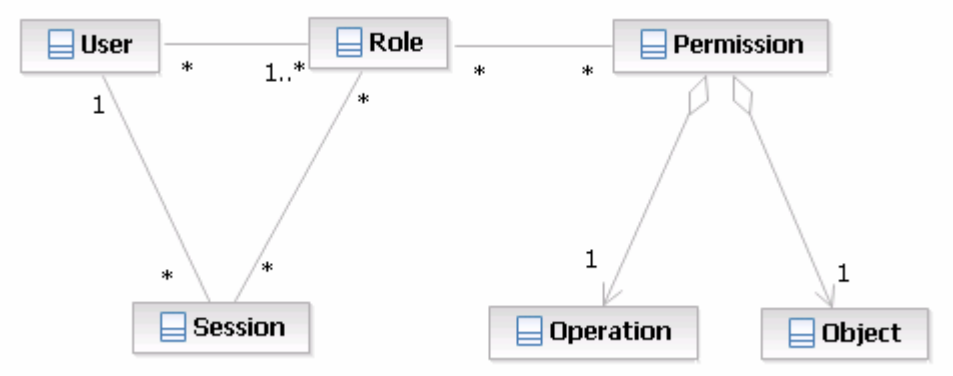
2.2.3 Szerepkör-alapú hozzáférés vezérlés (Role-based Access Control)

A szerepkör-alapú hozzáférés vezérlés öt fő entitása: a felhasználók, a szerepkörök, a műveletek, az objektumok és a munkamenetek. A felhasználókat szerepkörökhöz rendeljük. Egy felhasználóhoz több szerepkört, egy szerepkörhöz több felhasználót is rendelhetünk. A szerepekhez rendeljük a hozzáférési jogosultságokat, amelyek művelet-objektum párosokból állnak. Egy szerepkörhöz több jogosultság, egy jogosultság több szerepkörhöz is rendelhető. A felhasználókhoz munkamenetek rendelünk, egy felhasználónak természetesen több munkamenete is lehet, de egy munkamenet pontosan egy felhasználóhoz tartozik. Egy munkamenethez is rendelődnek szerepkörök, amelyek az aktuális felhasználó szerepkörei közül kerülnek ki, ezek az aktivált szerepkörök. A RBAC specifikáció nem szab megkötéseket a szerepkör-aktiválás folyamatára vonatkozóan, ez az implementáció hatáskörébe tartozik: az elterjedt implementációk az automatikus aktiválástól a felhasználó figyelmeztetésén át egészen az újrahitelesítésig folyamodhatnak új szerepkör aktiválásakor.

A felhasználó (amely szerepköreinek halmaza *k*) *B* műveletet *C* erőforráson akkor hajthat végre *D* munkamenet folyamán, ha a (*B, C*) páros szerepel *k* szerepkörök valamelyikéhez rendelve, és ezt a szerepkört a *D* munkamenet aktiválta. Vagyis engedélyezett kérés az, amelyhez tartozó művelet-erőforrás páros szerepel a kérést indító felhasználó szerepköreihez rendelt jogosultságok között, és a felhasználó munkamenete aktivált ilyen szerepkört.



5. ábra: A Role-based Access Control metamodellje



6. ábra: A Role-based Access Control UML metamodellje

Formálisan:

1. USERS, ROLES, OPS, OBS: felhasználók, szerepek, műveletek és objektumok.
2. UA $\subseteq$ USERS $\times$ ROLES, felhasználók és szerepek összerendelése (több-a-többhöz kapcsolat).
3. Egyazon szerepkörbe tartozó felhasználók:
   (r:ROLES) $\rightarrow 2^{USERS}$, assigned_users(r) = { u $\in$ USERS | (u, r) $\in$ UA }.
4. PRMS = $2^{(OPS \times OBS)}$, a jogosultságok készlete.
5. PA $\subseteq$ PRMS $\times$ ROLES, jogosultságok és szerepek összerendelése (több-a-többhöz kapcsolat).
6. Egy szerepkörhöz rendelt jogosultságok:
   (r: ROLES) $\rightarrow 2^{PRMS}$, assigned_permissions(r) = { p $\in$ PRMS | (p, r) $\in$ PA}.
7. SESSIONS: a munkamenetek készlete.
8. Egy felhasználó munkamenetei:
   (u: USERS) $\rightarrow 2^{SESSIONS}$, user_sessions(u) $\subseteq$ SESSIONS.
9. Egy munkamenethez tartozó szerepek:
   (s: SESSIONS) $\rightarrow 2^{ROLES}$, session_roles(s_i) $\subseteq$ { r $\in$ ROLES | (session_user(s_i), r) $\in$ UA}.
10. Egy felhasználó számára elérhető jogosultságok adott munkamenet során:
    session_perms(s:SESSIONS) $\rightarrow 2^{PRMS}$, $\bigcup_{r \in \text{ session\_roles(s)}}$ assigned_permissions(r)
11. Hozzáférés jogosultságának ellenőrzése:
    (s:SESSIONS, obj:OBS, op:OPS) $\rightarrow$ (TRUE | FALSE),
    check_permission(s, obj, op) = (
    (s$\in$ SESSIONS) ^

(obj ∈ OBS) ^

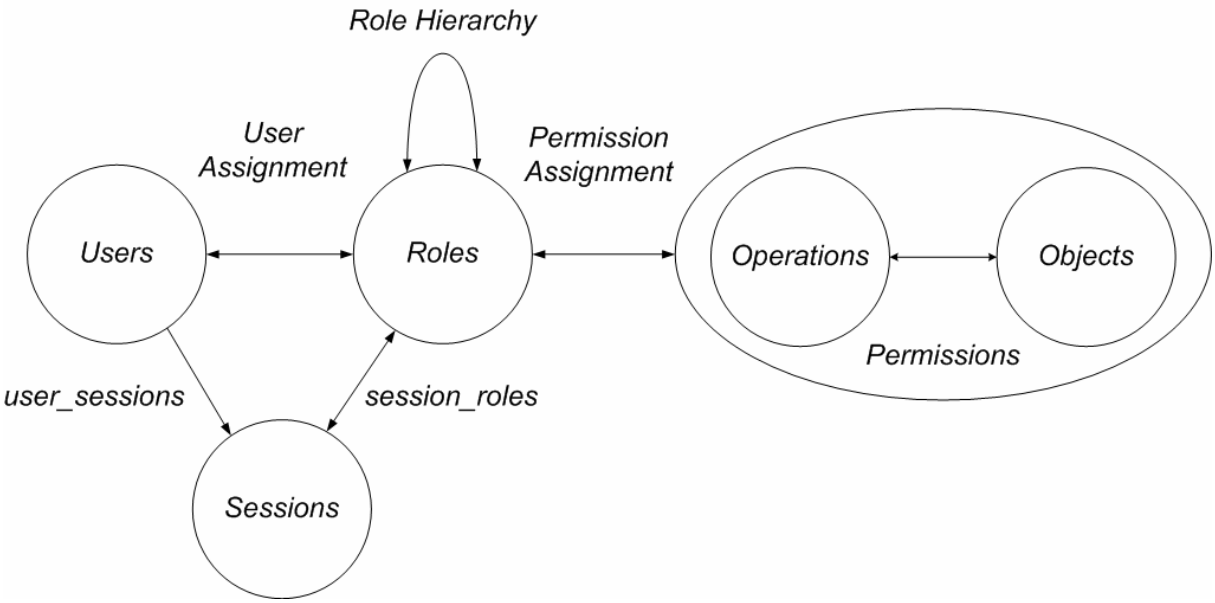
(op ∈ OPS) ^

(s ∈ user_sessions(u)) ^

(∃r∈ ROLES | r ∈ session_roles(s) ^ ((op, obj),r)∈ PA).

2.2.4 Hierarchikus, szerepkör-alapú hozzáférés vezérlés (Hierarchical Role-based Access Control)

A hierarchikus szerepkör-alapú hozzáférés vezérlés abban különbözik a hagyományostól, hogy szerepkörök hierarchiába szervezhetőek. Leíró ereje ugyanakkor megegyezik a hagyományos, szerepkör-alapú hozzáférés vezérlésével, mivel a szerepkörök kihajtogatásával a hagyományos RBAC-hoz juthatunk; különbséget csupán a kényelmesebb kezelhetőség jelent.



7. ábra: A Hierarchical Role-based Access Control metamodellje

2.3 Összegzés

A metamodell leírja az egyes modellek szintaktikáját (elemek készletét és azok kapcsolódási szabályait) valamint szemantikáját (hogy mit jelentenek az egyes elemek), ilyen módon meghatározza a modellek (példányok) megengedett szerkezetét és azok értelmezését. Ismertettem az elterjedt hozzáférés-vezérlési metamodelleket (DAC, MAC, RBAC, hRBAC), így ezen modellek strukturálási sajátosságait és értelmezését.

3 Az RBAC alkalmazása: a Tivoli Access Manager termékcsalád

Az egyes hozzáférés vezérlési módszerek közül a napjainkban legelterjedtebb a szerepkör alapú hozzáférés vezérlés, elsősorban mert jogosultsági modellje nagy számú felhasználó és komplex hozzáférési szabályok mellett is kompakt szabályrendszerrel történő megvalósítást tesz lehetővé. Az informatikai piacon számos vállalat rendelkezik ezen koncepciót megvalósító megoldással, melyek között az IBM megoldása a Tivoli Access Manager termékcsalád. Ezen fejezet ennek részletes ismertetésére összpontosít.

A termékcsalád működése általánosságban úgy írható le, hogy a hozzáférés vezérlést megvalósító komponens észleli a védett erőforráshoz érkező művelet kezdeményezését, a rendelkezésre álló házirendek és szabályrendszerek alapján meghozza a jogosultsági döntést, majd ettől függően továbbengedi a kérést az eredeti címzetthez, vagy visszautasítja azt.

A termékcsalád tagjainak közös eleme ezen jogosultság kiértékelő motor, amely egyúttal valamennyi termék alapját képezi. A jogosultság kiértékelés központi fogalma a *védett objektum tér*, amely virtuális és hierarchikus reprezentációja a védendő erőforrásoknak.

A TAM család egyes tagjai a különböző típusú feladatokra specializálódtak, így ennek megfelelően a következő erőforrástípusokat reprezentálhatják a védett objektumok:

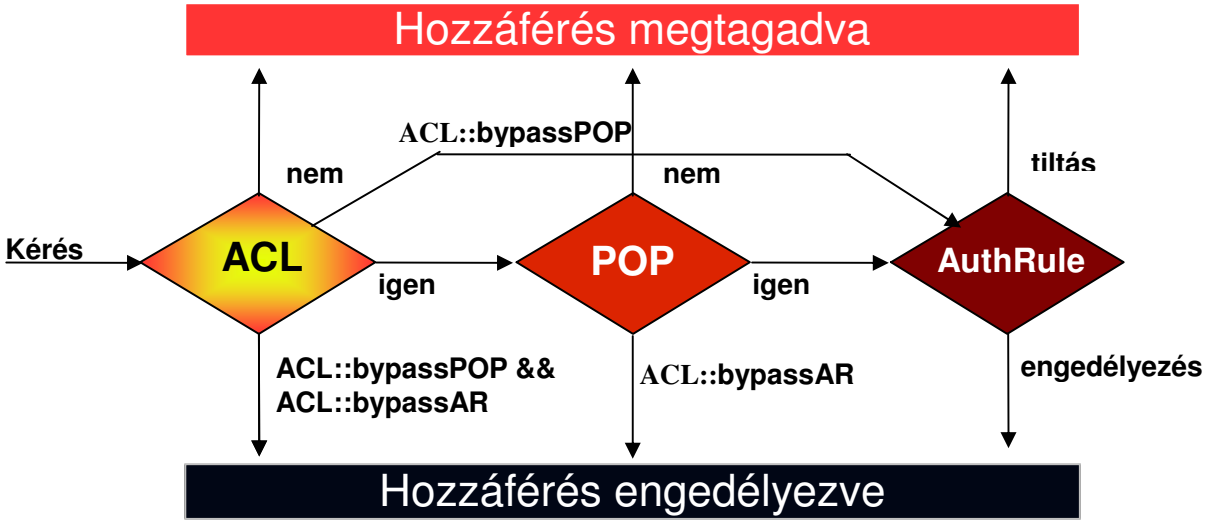
- Tivoli Access Manager for Operating Systems: nyomtatók, fájlok vagy operációs rendszer által nyújtott erőforrások és szolgáltatások
- Tivoli Access Manager for e-Business: webes alkalmazások URL-jei, J2EE vagy .NET környezetben az üzleti metódusok
- Tivoli Access Manager for Business Integration: üzenetsorok

A védett erőforrásokon értelmezett művelet lehet egy bejelentkezés, adminisztrátori ill. egyszerű felhasználói tevékenység, adat írás, olvasás, létrehozás vagy törlés.

A jogosultság kiértékelés három szinten történik:

1. *Hozzáférés vezérlési listák* (Access Control List, ACL) szintje: a hozzáférési szabályok első szintje; statikusan leírja, *ki-milyen erőforráson-milyen műveletet* jogosult ill. nem jogosult elvégezni.

2. *Védett erőforrás házirendek* (Protected Object Policy, POP) szintje: ezen a szinten nem lehet figyelembe venni a felhasználót és annak szerepköreit - szemben az ACL-lel - viszont megvalósíthatóak objektum-specifikus házirendek, például hogy adott védett webes erőforrást csak SSL-kapcsolaton belül, munkaidőben lehessen elérni..
3. *Authorizációs szabályok* (Authorizational Rule, AuthRule) szintje: a hozzáférési szabályok legáltalánosabb szintje, amely szövegesen definiálható szabályok mind az ACL-ek, mint a POP-ok számára rendelkezésre álló adatok alapján meghozzák a jogosultsági döntést.



8. ábra: A Tivoli Access Manager jogosultság kiértékelésének folyamata

A különböző szintű szabályok kiértékelése *lusta kiértékelés* (lazy evaluation) szerint történik: új kérés érkezésekor elsőként az ACL-ek kiértékelése történik meg. Amennyiben az ACL-ek kiértékelése pozitív eredménnyel zárul, vagyis a hozzáférés vezérlési listák alapján a kérés engedélyezhető, csak akkor értékelődnek ki a – jóval erőforrás-igényesebb – védett erőforrás házirendek, majd pedig ezek pozitív eredménye esetén az authorizációs szabályok. Bármelyik szint elutasító eredménye esetén a további szintek nem értékelődnek ki, a kérés visszautasításra kerül. Lehetőség van ugyanakkor – egy-egy speciális kapcsoló segítségével - az egyes szintek megkerülésére, ilyenkor a megkerült szintek (POP, AuthRule vagy mindkettő) nem kerülnek kiértékelésre.

3.1 Az RBAC megvalósítása: Tivoli Access Manager for e-Business

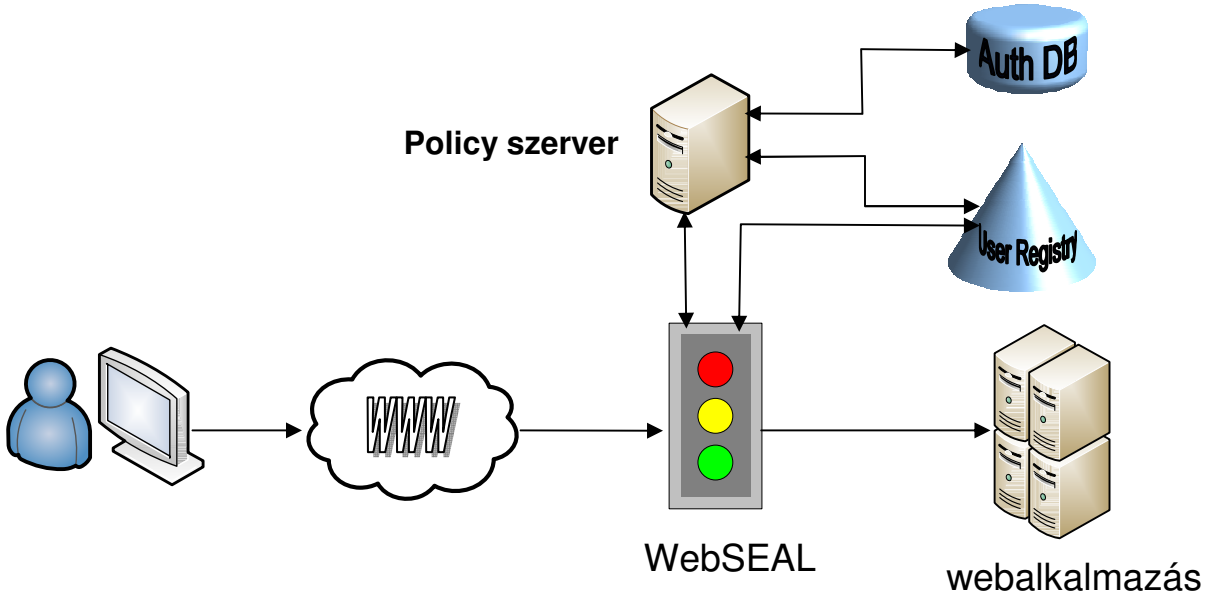
3.1.1 Funkcionális áttekintés, architektúra

Webes alkalmazások hozzáférés-vezérlésére szolgál a Tivoli Access Manager for e-Business megoldás. Segítségével a szervezetek kézben tarthatják az alkalmazások és adatok web-alapú hozzáférésének vezérlését. A megoldás egységes, házirend alapú hozzáférés-szabályzást valósít meg a szervezet webes rendszerein és alkalmazásain.

A megoldás lehetővé teszi meglévő alkalmazások hozzáférés-vezérléssel való ellátását, amelynek főbb komponensei a következők:

- A védendő alkalmazás
- Reverse proxy
- Policy Server
- Authorizációs címtár
- Felhasználói adatbázis

A megoldás standard működése a következőképp épül fel: a webes réteg (Internet vagy intranet) és a védendő alkalmazás közé épül be egy speciális fordított proxy, az ún. WebSEAL; a webalkalmazáshoz érkező kérések ezt követően ezen a proxyn kell, hogy áthaladjanak, így ezen a ponton lehetőség van a kérés authorizálására. A jogosultsági döntés meghozatala a felhasználói adatbázis és a releváns házirendek alapján történik. A felhasználó azonosítása a felhasználói címtár alapján történik, a házirendek lekérése pedig a Policy Serveren keresztül történik az authorizációs adatbázisból, amely Policy Server teszi lehetővé a házirendek karbantartását is.



9. ábra: A Tivoli Access Manager for e-Business architektúra

A jogosultsági döntés meghozatala tehát a WebSEAL-ben történik, itt történik az ACL-ek, a POP-ok és az AuthRule-k kiértékelése a korábban ismertetettek szerint. Pozitív döntés esetén a kérés továbbításra kerül a webalkalmazás felé, amely kiszolgálja a kérést, majd továbbítja az eredményt a proxy felé, amely ezután visszatér a kérés kezdeményezőjéhez. Negatív eredmény esetén azonban már a proxy visszautasítja a kérést, így a jogosulatlan kérések nem terhelik magát az erőforrást.

A szabályrendszer karbantartására, módosítására, valamint a védett erőforrásokhoz való hozzárendelésére intuitív, központi webes felületen keresztül van lehetőség.

Mivel a WebSeal lokálisan egy gyorsítótárban is tárolja a releváns házirendeket, ezért a Policy Server esetleges meghibásodása nem okoz kiesést az alkalmazás elérhetőségére nézve, csupán a házirendek módosulásai nem jutnak érvényre a Policy Server működésének helyreállásáig. A WebSeal viszont ilyen értelemben egyszeres hibapontnak tekinthető, kiesése esetén az alkalmazás elérhetetlenné válhat, ezért ezt a komponenst magas rendelkezésre állásúvá szokás tenni.

3.1.2 Kiterjesztési lehetőségek

Meglévő alkalmazások védelmének biztosításán kívül a Tivoli Access Manager API[13] felülete segítségével lehetőség van az újonnan készülő alkalmazások hozzáférés-szabályzásának központosított megvalósítására is oly módon, hogy az alkalmazás kódjában csupán a döntési pontot kell elhelyezni, magát a döntést a központi, nagy rendelkezésre állású jogosultságellenőrző motor végzi.

Ez akkor rendkívül előnyös, ha szabványos, deklaratív biztonsági megoldás nem áll rendelkezésre (pl. C, C++, .NET programozási nyelvek esetén), vagy ha a szabványos megoldás (ilyen J2EE esetén a Java Authentication and Authorization Service (JAAS) vagy a Java Authorization Contract for Containers (JACC)) leíróereje nem elég, mert azon túlmutató, például környezetfüggő paraméterek figyelembe vételére van szükség.

Ezen megoldás használatával tehát lehetővé válik az alkalmazások jogosultságkezelésének átkonfigurálása csupán a kihelyezett hozzáférési házirendek megfelelő módosításával; elkerülhetővé válik, hogy a jogosultsági változások esetén – ha maga a döntési pont nem, csak a döntési szabályok változnak - az alkalmazás kódján kelljen módosítani. Tehát ha pl. egy újabb csoportnak szeretnénk jogot biztosítani az alkalmazás egy adott funkciójához, akkor az alkalmazás módosítása, újrafordítása, tesztelése, stb. helyett elegendő csupán a releváns házirend kiegészítése egy új bejegyzéssel, mellyel rengeteg idő és költség takarítható meg.

Ezen API hívásra épülő megoldás korlátja, hogy az alapkoncepció esetén nincs lehetőség futás idejű, dinamikusan változó paraméterek kiértékelésére, csupán előre definiált statikus szabályok érvényre juttatására.

Mivel azonban a Tivoli Access Manager API felülete lehetőséget biztosít egyedi bővítmények fejlesztésére is, a csupán statikus hozzáférési szabályokat kezelni képes jogosultsági rendszer kiváltható egy, immár kontextusfüggő paramétereket is figyelembe venni képes jogosultsági modellel: eme korlátozást hívatott feloldani a DynRBAC és az azt implementáló AuthEngine [2].

[13] Application Programming Interface

4 Dinamikus, szerepkör alapú hozzáférés vezérlési szolgáltatás: DynRBAC

A hagyományos szerepkör-alapú hozzáférés-vezérlés segítségével meghatározható, hogy *ki – milyen műveletet* végezhet *mely védett erőforrásokon*. Az esetek döntő többségében a jogosultsági döntés teljes egészében meghozható ezen paraméterek alapján, azonban előfordulnak olyan speciális igények, üzleti logikai vagy egyéb megszorítások, amelyek szükségessé teszik bizonyos kontextusfüggő paraméterek figyelembe vételét az authorizációs szabályok kiértékelésekor.

A DynRBAC [2] egy olyan hozzáférés-vezérlési modell, ami kiterjeszti a szerepkör-alapú hozzáférés-vezérlést (RBAC) a jogosultsági döntés során kontextusfüggő paraméterek figyelembe vételének képességével azáltal, hogy kiegészíti a jogosultság elbírálási döntést őrfeltételek kiértékelésének a lehetőségével. (Az őrfeltétel olyan logikai kifejezés, amely a hívási kontextus adataira hivatkozva megadja azokat a feltételeket, amik a hozzáféréshez engedélyezéséhez szükségesek.) Ezáltal a gyártóspecifikus, nem szabványos hozzáférés vezérlési kiterjesztések is egységesen reprezentálhatóvá, validálhatóvá válnak a modell szintjén.

Az őrfeltételek által hivatkozott paraméterek számára és típusára vonatkozóan a modell nem tesz megkötést, melynek révén lényegében tetszőleges hozzáférési szabály megfogalmazható.

4.1 AuthEngine: a DynRBAC minta-implementációja

Az AuthEngine - a DynRBAC modellt implementáló Tivoli Access Manager alapú bővítmény - felruházza a TAM infrastruktúrát azzal a képességgel, hogy speciális, környezeti paramétereket figyelembe venni képes biztonsági házirendeket is betartasson.

A hozzáférési kérelem kiértékelése során lehetőség nyílik arra, hogy a szabálymotor a védett alkalmazás kontextusából származó változókat is bevonja az authorizációs döntési folyamatba. A paraméterek begyűjtését és átadását egy kiterjesztett programozói felület teszi lehetővé. Az API hívások alkalmazásokba való beépítésével elkerülhető az, hogy az authorizációs döntés folyamatát az alkalmazás

kódjában kelljen megvalósítani. Így egy olyan externalizált, központilag menedzselhető biztonsági szolgáltatást kapunk, amely lehetővé teszi, hogy a hozzáférési szabályrendszer módosítása esetén sem alkalmazáskód-módosítás, sem az alkalmazások újrakonfigurálására, sem pedig újraindításuk ne váljon szükségessé, hiszen azok néhány másodperc alatt automatikusan érvénybe lépnek.

Bár a Tivoli Access Manager jogosultsági motorja támogatja a dinamikus szabályok használatát (AuthRule), az ilyen szabályok kiértékelése nagy terhelést jelent a jogosultság-motor számára, olyan nagy terhet, ami egy masszívan párhuzamos alkalmazás esetében nem jelent működőképes alternatívát. Az AuthEngine kiterjesztésnek elsődleges célja az volt, hogy a logikai szabályok kiértékelését olyan módon tegye lehetővé, hogy az akár a pénzügyi szektor web alapú alkalmazásai esetében is megállja helyét.

Maga az implementáció olyan módon használja a TAM authorizációs adatbázisát - ami minden esetben a jogosultsági döntés helyére replikálódik - hogy előre kiszámítva ott tárolja a kifejezések igazság-tábláját. Ez a megközelítés lehetővé tette, hogy natív sebességgel, overhead nélkül történjen a kontextusfüggő jogosultsági kérések kiértékelése, viszont technikai korlátot jelent a kontextus paraméterek típusa és száma terén. Vegyük észre, hogy a paraméterek számának növelésével a kontextusfüggő szabály igazságtáblájának mérete exponenciálisan nő, így ez erősen peremfeltételeket szab egy kifejezés komplexitására.

Az igazságtáblák méretének exponenciális robbanása úgy kerülhető el, hogy bizonyos függőségek ill. függetlenségek kihasználásával funkcionális dekompozíciót alkalmazva kisebb, jelen implementációban legfeljebb ötös csoportokra oszthatóak a figyelembe veendő kontextusfüggő paraméterek. Ezen ötös csoportok logikai összekapcsolásával (fa- vagy lánc-struktúrába szervezésével) lehetőség nyílik komplexebb szabályrendszerek megfogalmazására.

Fontos megjegyezni, hogy a kontextusfüggő paraméterek legfeljebb ötös csoportokba szervezése nem DynRBAC modellből, pusztán az azt megvalósító AuthEngine implementációjából adódó korlát, ott is pusztán technikai okai vannak ezen korlátozás bevezetésének. Maga a modell tetszőleges számú paraméter figyelembe vételét lehetővé teszi. Az implementáció ezen túlmenően bináris paramétereket használ, amely azonban nem korlátozza a modell által nyújtott leíróerőt.

Az DynRBAC működését demonstráló AuthEngine implementációban a szabályrendszer karbantartására egy alternatív webes felület szolgál, amely a nagyobb rugalmasság érdekében a szabályok szöveges formában (logikai kifejezésekkel) történő szerkesztését is támogatja. Az így létrehozott szabályok lehetővé teszik a komplex, egyedi igényeket tartalmazó biztonsági házirendek megvalósítását a magas performancia megtartása mellett.

A plug-in tehát rugalmas és jól skálázható jogosultsági szabálykiértékelést nyújt Java technológián alapuló alkalmazások számára az informatikai rendszer performanciájának érzékelhető csökkenése nélkül.

5 A DynRBAC modellje

A DynRBAC a hagyományos Role-based Access Control kiterjesztése, így absztrakt modellje a hagyományos RBAC-on alapul, a dinamikus viselkedést lehetővé tevő elemekkel kiegészítve azt.

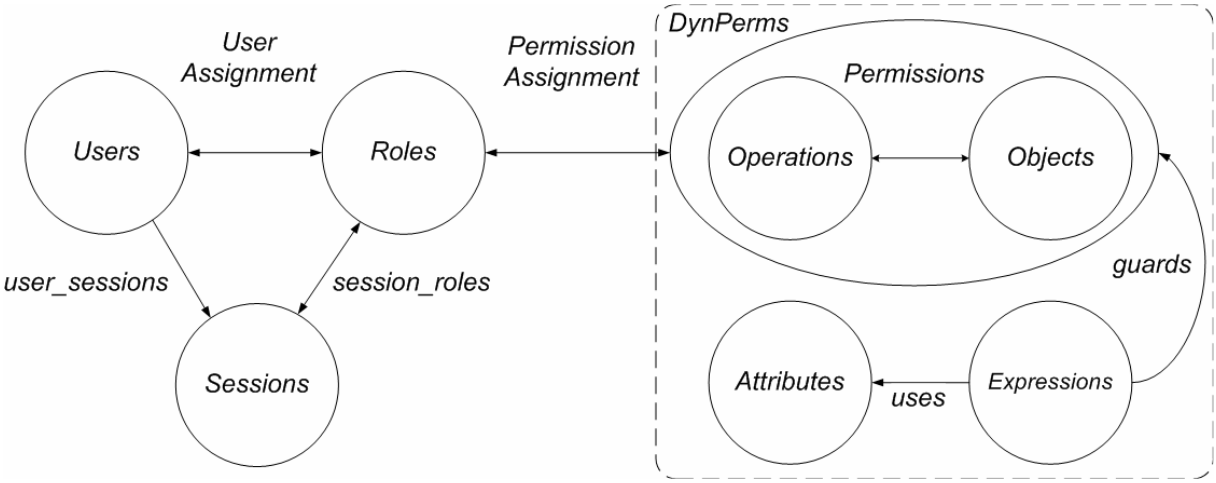
Az alap RBAC entitásain (*felhasználók*, *szerepek*, *műveletek*, *objektumok* és *munkamenetek*) kívül két új entitás jelenik meg: az *attribútumok*, amelyek a futásidejű, kontextusfüggő paraméterek reprezentációi, valamint *kifejezések*, amelyek a hozzáférés engedélyezett attribútumérték-kombinációit írják le. Attribútum lehet bármely olyan, a döntés kontextusából származó paraméter, amely szükséges lehet a döntés meghozatalához; egy internetbankos tranzakció esetén például, hogy a kérést a bank nyitvatartási ideje alatt indították-e, vagy 100.000Ft alatti tranzakcióról van-e szó. A logikai kifejezések az attribútumok engedélyezett kombinációi, például „a bank nyitvatartási ideje alatt, 100.000Ft felett”, vagy „a nyitvatartási időn kívül, csak 100.000Ft alatt”.

Ezen új entitások a *jogosultság* (permission) entitáson keresztül kapcsolódnak a modellhez: a DynRBAC modelljében a jogosultság entitásokat az RBAC-hoz hasonlóan művelet-objektum párosok alkotják, ám az entitáshoz őrfeltételként kapcsolódik egy kifejezés is. A kifejezés a futásidejű attribútumok értéke határozza meg, hogy az adott jogosultág engedélyezett-e. Az objektumok, műveletek és kifejezések (amelyek attribútumokat használnak) alkotta jogosultságokat *dinamikus jogosultságok*nak nevezzük. Ezen belül az egyes entitások a következőképp kapcsolódnak egymáshoz: az

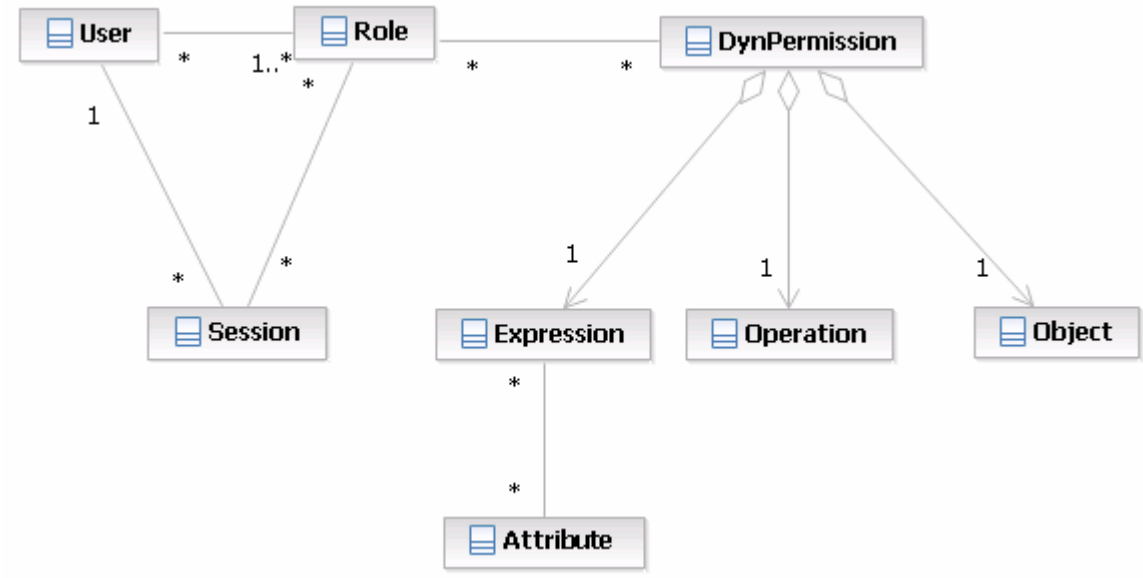
egyes objektumokon műveleteket értelmezünk, a művelet végrehajtási kontextusa futásidejű attribútumokat tartalmaz; a logikai kifejezések pedig az attribútumok azon értékeinek kombinációit határozzák meg, amelyek mellett az adott objektumon végzett adott műveletek engedélyezettek. Egy objektumon több műveletet is értelmezhetünk. Egy művelet végrehajtási kontextusa több attribúmot is szolgáltathat, ugyanakkor egy attribútum több művelet szempontjából is releváns lehet. Egy logikai kifejezés több attribútumot is használhat, valamint egy attribútum több logikai kifejezésnek is része lehet. Egy objektum-művelet párhoz kifejezések rendelhetők. Ilyen módon egy dinamikus jogosultság lehet például egy internetbankos átutalás indítása munkaidőn kívül, 100.000Ft feletti összeggel.

Az ilyen dinamikus jogosultságokat az RBAC-hoz hasonlóan szerepkörökhöz rendeljük. Minden felhasználóhoz szerepkörök egy halmaza van rendelve. A felhasználó az egyes munkamenetek során ebből a halmazból aktiválhat szerepköröket, mellyel a munkamenet elnyeri az aktivált szerepkörökhoz tartozó dinamikus jogosultságokat.

A felhasználó (amely aktivált szerepköreinek halmaza *k*) *B* műveletet *C* erőforráson akkor hajthat végre, ha a (*B*, *C*) pároshoz csatolt *A* felhasználóra (a *k* aktivált szerepkörein keresztül) vonatkozó *E* logikai kifejezés által igényelt (*p1*, *p2*, …) paraméterhalmaz rendelkezésre áll a hívási kontextusban, és a paraméterek aktuális értékével kiértékelt *E* logikai kifejezés teljesül, azaz igaz a visszatérési értéke. Vagyis engedélyezett kérés az, amelyhez tartozó művelet-erőforrás pároshoz kapcsolt, adott szerepkörhöz tartozó logikai kifejezés a környezeti paraméterek aktuális értéke mellett igaz értéket ad.



10. ábra: A Dynamic Role-based Access Control metamodellje



11. ábra: A Dynamic Role-based Access Control UML metamodellje

Formálisan:

1. USERS, ROLES, OPS, OBS: felhasználók, szerepek, műveletek és objektumok.
2. UA $\subseteq$ USERS $\times$ ROLES, felhasználók és szerepek összerendelése (több-a-többhöz kapcsolat).
3. Egyazon szerepkörbe tartozó felhasználók:
   (r:ROLES) $\rightarrow 2^{\text{USERS}}$, assigned_users(r) = { u $\in$ USERS | (u, r) $\in$ UA }.
4. ATTRS: a kontextusfüggő attribútumok készlete.
5. EXPRS: a logikai kifejezések készlete.
6. AA $\subseteq$ ATTRS $\times$ EXPRS, attribútumok és logikai kifejezések összerendelése (több-a-többhöz kapcsolat).

7. Egy kifejezés által használt attribútumok:
   (e:EXPRS) $\rightarrow 2^{ATTRS}$, used_attributes(e) = { a $\in$ ATTRS | (a, e) $\in$ AA }.
8. DynPERMS = $2^{(OPS \times OBS \times EXPRS)}$, a jogosultságok készlete.
9. Jogosultsághoz kapcsolt logikai kifejezés kielégítése:
   satisfies(p, att): (p$\in$ DynPERMS, att$\in$ ATTRS) $\rightarrow$ (TRUE | FALSE).
10. PA $\subseteq$ DynPERMS $\times$ ROLES, jogosultságok és szerepek összerendelése (több-a-többhöz kapcsolat).
11. Egy szerepkörhöz rendelt jogosultságok:
    (r: ROLES) $\rightarrow 2^{PRMS}$, assigned_permissions(r) = { p $\in$ DynPERMS | (p, r) $\in$ PA}.
12. SESSIONS: a munkamenetek készlete.
13. Egy felhasználó munkamenetei:
    (u: USERS) $\rightarrow 2^{SESSIONS}$, user_sessions(u) $\subseteq$ SESSIONS.
14. Egy munkamenethez tartozó szerepek:
    (s: SESSIONS) $\rightarrow 2^{ROLES}$, session_roles(s_i) $\subseteq$ { r $\in$ ROLES | (session_users(s_i), r) $\in$ UA}.
15. Egy felhasználó számára elérhető jogosultságok adott munkamenet során:
    session_perms(s:SESSIONS) $\rightarrow 2^{DynPERMS}$, $\bigcup_{r \in \text{ session_roles(s)}}$assigned_permissions(r)
16. Hozzáférés jogosultságának ellenőrzése:
    (s:SESSIONS, obj:OBS, op:OPS, att:ATTRS) $\rightarrow$ (TRUE | FALSE),
    check_permission(s, obj, op, att) = (
        (s$\in$ SESSIONS) ^
        (obj $\in$OBS) ^
        (op $\in$ OPS) ^
        (s $\in$ user_sessions(u)) ^
        (att $\supseteq$ used_attributes(e) | e $\in$ EXPRS ^ (p = (op, obj, e)$\in$ DynPERMS)) ^
        ($\exists$ r$\in$ ROLES | r $\in$ session_roles(s) ^ (p, r)$\in$ PA ^ satisfies(p, att)).

6 A DynRBAC szimulációja SAL környezetben

A valós életbeli, kellően komplex szabályrendszerek esetén a megfogalmazott kritériumok teljesülésének vizsgálata hagyományos módszerekkel, azaz a szabályrendszer modellezése nélkül kivitelezhetetlen feladat.

A DynRBAC modell lehetőséget biztosít a jogosultsági döntés során a művelet kontextusából származó attribútumokon alapuló szabályok figyelembe vételére, mégpedig modellezhető és validálható módon, melyre a korábbi megoldások nem adtak lehetőséget.

Ezen lehetőséggel élve szimulációt és modellellenőrzést végzek a DynRBAC modellt megvalósító AuthEngine mechanizmusainak demonstrálására: az AuthEngine működését leképezem egy modellellenőrző eszköz nyelvére, szimulálom a jogosultság kiértékelés folyamatát egy adott modellen, majd pedig biztonsági kényszerek, invariánsok teljesülését vizsgálom azon.

A szimuláció segítségével az AuthEngine entitásait, jogosultság-kiértékelésének folyamatát szemléltetem, a modellellenőrzés segítségével pedig olyan újrahasználható kritériumok teljesülését ellenőrzöm, amelyek jellemzőek az adott szabályrendszer megkerülhetetlenségére.

6.1 A szimulációs környezet

A dinamikus, szerepkör-alapú hozzáférés-vezérlést megvalósító AuthEngine szimulációjának elvégzése két kézenfekvő alternatíva kínálkozott: a korlátprogramozást lehetővé tevő Sicstus Prolog[14], valamint a formális analízist támogató SAL[15] alkalmazás használata. Végül a SAL szoftverének legújabb, 3.0-s verzióját választottam, elsősorban széleskörű lehetőségeket biztosító leíró nyelve, szimulációs és modellellenőrzési képességei miatt.

A szimulációs környezet egy 2.16 GHz-es 2 magos Intel processzoros, 3 GB RAM-mal rendelkező notebook volt, Windows XP SP2 operációs rendszerrel. Mivel a

[14] http://www.sics.se/sicstus.html
[15] Symbolic Analysis Laboratory, http://www.csl.sri.com/projects/sal/, http://sal.csl.sri.com/

SAL Linux/Unix környezetre íródott, emiatt Windows környezetben Unix-emulátor használatával futtatható, amelyre a Cygwin[16] 2.573 változatát használtam.

6.2 Az AuthEngine elemeinek SAL modellbeli reprezentációja

A SAL modell felállításakor az AuthEngine, illetve a Tivoli Access Manager fogalomkészletének nem teljes halmazát vettem alapul, hogy a modell mérete kezelhető maradjon. Ugyanakkor törekedtem arra, hogy a modellezett elemek lefedjék a DynRBAC fogalmainak, illetve az AuthEngine jogosultság kiértékelő motorjának működését a lehető legteljesebb mértékben.

Ezen megfontolások alapján a SAL modellben a következő entitásokat képeztem le:

- egyedi felhasználóra vonatkozó *ACL*-ek
- csoportra vonatkozó *ACL*-ek
- a kontextusfüggő attribútumok készletét reprezentáló *ConditionGroup*-ok.

Mivel a fennmaradó elemek (*Protected Object Policy-k* és *Authentication Rule*-k) a Tivoli Access Manager gyártóspecifikus kiterjesztései - amik helyettesíthetők DynRBAC kifejezésekkel - a szimulátor SAL modellje ezeket nem tartalmazza.

Bár a SAL alapvetően Petri-hálók modellezésére és szimulációjára szolgál, leíró nyelve lehetővé tesz imperatív programozási elemek (változók, tömbök, if-then-else szerkezetek) használatát is. A modell a SAL eme mindkét tulajdonságát kihasználva készült el.

6.2.1 Az Access Control List-ek

A felállított modellben a felhasználói, illetve csoportra vonatkozó ACL-ek kerültek megvalósításra. Egy ACL bejegyzés egy hozzáférési szabályt határoz meg: *ki / milyen csoport*, *milyen erőforráson*, *milyen műveletet* végezhet el.

Az ACL-ek modellezése során a SAL Petri-háló működését vizsgáló lehetőségét használtam ki: az egyes ACL bejegyzéseket, mint konkurrens módon kiértékelendő tranzíciókat képeztem le. Egy ilyen tranzíció akkor tud tüzelni, ha a bemeneti

[16] http://www.cygwin.com/

paraméterek megfelelnek az adott ACL egy bejegyzésének, és ilyenkor a döntést pozitívra állítják be.

Definíciószerűen, amennyiben a Tivoli Access Manager ACLjei közül az adott kérésre vonatkozóan egy is megengedő bejegyzést tartalmaz, úgy a jogosultsági döntés eredménye engedélyezés lesz. Ennek megfelelően, ha az egyes ACL bejegyzéseknek megfelelő tranzíciók közül legalább egy tüzelni tud, úgy a döntés eredménye engedélyezés lesz. Ha egyetlen ACL bejegyzésnek megfelelő tranzíció sem tud tüzelni, azaz egyetlen, az adott kérést megengedő ACL bejegyzés sincsen, úgy a döntés később, a ConditionGroup-ok alapján születik meg.

Egy felhasználói ACL bejegyzés modellbeli reprezentációja a következő:

```
TRANSITION
     [
     ...
     []
      <<ACLnév>>:
          (myname = <<felhasználónév>> AND
          resource = <<erőforrásnév>> AND
          operation = <<műveletnév>>) --> decision'= TRUE
     ...
     ]
```

amely tehát pontosan egy (<<*felhasználónév*>>, <<*erőforrásnév*>>, <<*műveletnév*>>) ACL bejegyzésnek felel meg, vagyis azt jelenti, hogy ha <<*felhasználónév*>> <<*erőforrásnév*>>-n <<*műveletnév*>>-t kezdeményez, akkor a kérés engedélyezésre kerül.

Egy csoport ACL bejegyzés modellbeli reprezentációja pedig a következő:

```
TRANSITION
     [
     ...
     []
      <<ACLnév>>:
          (EXISTS (i : Index): mygroups[i] = <<csoportnév>>
AND
          resource = <<erőforrásnév>> AND
          operation = <<műveletnév>>) --> decision'= TRUE
     ...
     ]
```

amely tehát pontosan egy (<<*csoportnév*>>, <<*erőforrásnév*>>, <<*műveletnév*>>) ACL bejegyzésnek felel meg, vagyis azt jelenti, hogy ha egy <<*csoportnév*>>-beli

felhasználó <<*erőforrásnév*>>-n <<*műveletnév*>>-t kezdeményez, akkor a kérés engedélyezésre kerül.

6.2.2 A ConditionGroup-ok

A felállított modellben a kontextusfüggő paraméterek az AuthEngine által is használt ConditionGroup-ok formájában kerültek megvalósításra. Egy ConditionGroup 5 attribútum együttesét jelenti, amelyek felhasználó-objektum-művelet hármashoz kapcsolásakor megadhatóak az attribútumok azon kombinációi, amelyek esetén a kérés engedélyezhető.

A ConditionGroup definiálása a modellben a következőképp történik:

```
ConditionGroups: TYPE [0 .. 4] OF BOOLEAN;
```

Az eredeti DynRBAC-t megvalósító AuthEngine implementáció ötös BOOLEAN csoportokat használ, de látható, hogy ezen a ponton megadható az attribútumcsoport mérete és típusa is, hiszen maga a DynRBAC modell megengedi tetszőleges számú és típusú paraméter figyelembe vételét.

A Tivoli Access Manager authorizációs adatbázisából felolvasott ConditionGroup-ok deklarálása ezután a következő formában történik:

```
GLOBAL <<conditionGroupNév>> : ConditionGroups
```

A felhasználó-objektum-művelet hármashoz kapcsolt ConditionGroup-ok a modellben egy-egy konkurrensen kiértékelendő tranzícióként reprezentáltak: amennyiben a kérés és a környezeti attribútumok bármely hozzákapcsolt ConditionGroup mintájára illeszkednek, a neki megfelelő tranzíció tüzelni tud, és a döntést pozitívra állítja. Amennyiben egyik ConditionGroup-ra (és ACL-re) sem illeszkednek az adott kérés paraméterei, úgy a döntés elutasítás lesz.

Egy, felhasználóhoz, objektumhoz és művelethez kapcsolt ConditionGroup modellbeli reprezentációja a következő:

```
TRANSITION
     [
     ...
     []
     <<usedConditionGroup_engedélyezettKombinációNév>>:
          (resource = <<erőforrásnév>> AND
           operation = <<műveletnév>> AND
           myname = <<felhasználónév>> AND
```

```
            <<conditionGroupNév>>[0] = [TRUE | FALSE] AND
            <<conditionGroupNév>>[1] = [TRUE | FALSE] AND
            <<conditionGroupNév>>[2] = [TRUE | FALSE] AND
            <<conditionGroupNév>>[3] = [TRUE | FALSE] AND
            <<conditionGroupNév>>[4] = [TRUE | FALSE])
            --> decision' = TRUE
      ...
      ]
```

amely tehát azt jelenti, hogy ha a <<*felhasználónév*>> <<*erőforrásnév*>>-n <<*műveletnév*>>-t kezdeményez, és a <<*conditionGroupNév*>> ConditionGroup attribútumainak értéke a megadottnak megfelel, akkor a kérés engedélyezésre kerül.

Ha egyik ConditionGroup-ra (és ACL-re) sem illeszkednek az adott kérés paraméterei, a kérés elutasításra kerül:

```
TRANSITION
      [
      ...
      []
      ELSE --> decision'= FALSE
      ...
      ]
```

6.2.3 A bemeneti és kimeneti paraméterek

A modell bemeneti paraméterei a kérést indító felhasználó neve és csoporttagságai, az igényelt erőforrás illetve művelet, valamint a kontextusfüggő attribútumok aktuális értékei. Hogy ezen bemeneti paraméterek milyen értékeket vehetnek fel, az a Tivoli Access Manager authorizációs adatbázisából kerül felolvasásra:

```
Names: TYPE = {<<felolvasott felhasználónevek vesszővel elválasztva>>}
Groups: TYPE = {<<felolvasott csoportok vesszővel elválasztva>>}
Resources: TYPE = {<<felolvasott erőforrások vesszővel elválasztva>>}
Operations: TYPE = {<<felolvasott műveletek vesszővel elválasztva>>}
```

A bemeneti paramétereket ezután globális változókként deklaráljuk a modellben, az alábbi formátumban:

```
GLOBAL myname : Names
GLOBAL mygroups : ARRAY [0 .. N] OF Groups
GLOBAL resource : Resources
GLOBAL operation: Operations
...
INITIALIZATION
...
       myname = <<a kérést indító neve>>
```

```
        mygroups[0] = <<a kérést indító csoportja>>
        mygroups[1] = <<a kérést indító csoportja>>
        ...
        resource = <<az igényelt erőforrás>>
        operation = <<az igényelt művelet>>

        <<conditionGroupNév1>>[0] = [TRUE | FALSE]
        <<conditionGroupNév1>>[1] = [TRUE | FALSE]
        <<conditionGroupNév1>>[2] = [TRUE | FALSE]
        <<conditionGroupNév1>>[3] = [TRUE | FALSE]
        <<conditionGroupNév1>>[4] = [TRUE | FALSE]

        <<conditionGroupNév2>>[0] = [TRUE | FALSE]
        <<conditionGroupNév2>>[1] = [TRUE | FALSE]
        <<conditionGroupNév2>>[2] = [TRUE | FALSE]
        <<conditionGroupNév2>>[3] = [TRUE | FALSE]
        <<conditionGroupNév2>>[4] = [TRUE | FALSE]
        ...
...
```

A modell kimenete egyetlen igaz/hamis értékkészletű paraméter, a döntés eredménye, amelyet a következő módon definiálunk:

```
OUTPUT decision : BOOLEAN
...
INITIALIZATION
...
        decision = FALSE;
...
```

6.3 A szimuláció

Az elkészült SAL modell szimulációja során egy, a Tivoli Access Manager for e-Business 6.1 bővítményeként működő AuthEngine authorizációs adatbázisából felolvasott hozzáférési szabályokat vettem alapul (lásd: Függelék). A szimuláció elvégzéséhez a SAL rendszerhez tartozó `sal-sim` nevű eszközt használtam.

A szimuláció során a SAL kódban leírt hozzáférési szabályokat nem, csupán a bemeneti paramétereket (a kérést indító felhasználó neve és csoporttagságai, az igényelt erőforrás illetve művelet, valamint a kontextusfüggő attribútumok értékei) változtattam, majd a modell szimulációjának futtatása után a kimeneti paraméter értékét, vagyis a döntés eredményét kísértem figyelemmel.

A SAL ugyanakkor nem követeli meg valamennyi bemeneti paraméter értékének megadását: lehetőség van egy vagy több bemeneti paraméter értékét lekötetlenül hagyni. Ilyenkor a szimuláció során a SAL motorja a lekötetlen paraméterek összes lehetséges értékét figyelembe véve (a megadott értékkészlet alapján) értékeli ki a tranzíciókat, vagyis a hozzáférési szabályokat, és jeleníti meg az eredmény értékét a paraméterek egyes lekötése esetén. Ez egy rendkívül hasznos lehetőség a szimuláció végzése során, figyelembe kell venni ugyanakkor, hogy amíg egy bináris paraméter szabadon hagyása esetén két eredmény születhet, addig például két bináris és két háromértékű paraméter szabadon hagyása esetén már 36 féle.

1. eset: felhasználói ACL alapján engedélyezendő kérés

Az első szimuláció során olyan bemeneti paramétereket határoztam meg, amelyek a következő felhasználói ACL alapján engedélyező döntést kell, hogy eredményezzenek:

```
TRANSITION
     [
     ...
     []
      useracl1:
            (myname = venus AND
             resource = Szef AND
             operation = Access) --> decision'= TRUE
     ...
     ]
```

A szimuláció eredménye:

```
sal > (display-curr-states)
```

```
State 1
--- System Variables (assignments) ---
myname = venus
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = true
----------------------------
```

2. eset: csoport ACL alapján engedélyezendő kérés

A második szimuláció során olyan bemeneti paramétereket határoztam meg, amelyek az alábbi csoporttagságra vonatkozó ACL alapján engedélyező döntést kell, hogy eredményezzenek:

```
TRANSITION
     [
     ...
     []
      groupacl1:
           (EXISTS (i : Index): mygroups[i] = Admin AND
            resource = Szef AND
            operation = Access) --> decision'= TRUE
     ]
```

A szimuláció eredménye:

```
sal > (display-curr-states)
```

```
State 1
--- System Variables (assignments) ---
myname = sec_master
mygroups[0] = User
mygroups[1] = Admin
```

```
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = true
----------------------------
```

3. eset: ConditionGroup alapján engedélyezendő kérés

A következő szimuláció során olyan bemeneti paramétereket határoztam meg, amelyek az alábbi ConditionGroup alapján engedélyező döntést kell, hogy eredményezzenek:

```
TRANSITION
     [
     ...
     []
     usedconditiongroup_2_authorizedcombination_1:
          (resource = Szef AND
           operation = Access AND
           myname = mars AND
           transProperties[0] = true AND
           transProperties[1] = false AND
           transProperties[2] = true AND
           transProperties[3] = false AND
           transProperties[4] = true) --> decision' = TRUE
     ...
     ]
```

A szimuláció eredménye:

```
sal > (display-curr-states)
```

```
State 1
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
```

```
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = true
----------------------------
```

4. eset: elutasítandó kérés

A következő szimuláció során olyan bemeneti paramétereket határoztam meg, amelyek sem a felhasználói és csoporttagságra vonatkozó ACL-ek, sem pedig a ConditionGroup-ok alapján nem engedélyezett döntéshez vezetnek. A szimuláció eredménye:

```
sal > (display-curr-states)
State 1
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = false
transProperties[3] = false
transProperties[4] = false
decision = false
----------------------------
```

5. eset: felhasználónév alapján eldöntendő kérés

A következő szimuláció folyamán nem határoztam meg a `myname` paraméter értékét. A többi beállított paraméter értéke mellett *sec_master* számára engedélyezendő a kérés, a többi felhasználó számára elutasítandó. A szimuláció eredménye:

```
sal > (display-curr-states)
State 1
--- System Variables (assignments) ---
```

```
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = false
transProperties[3] = false
transProperties[4] = false
decision = false
-----------------------------
State 2
--- System Variables (assignments) ---
myname = sec_master
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = false
transProperties[3] = false
transProperties[4] = false
decision = true
-----------------------------
State 3
--- System Variables (assignments) ---
myname = venus
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
```

```
transProperties[1] = false
transProperties[2] = false
transProperties[3] = false
transProperties[4] = false
decision = false
----------------------------
```

6. eset: ConditionGroup egy paramétere alapján eldöntendő kérés

A következő szimuláció folyamán nem határoztam meg a `transProperties` nevű ConditionGroup első paraméterének értékét. A többi beállított paraméter értéke mellett *true* érték esetén engedélyezendő, *false* esetén elutasítandó a kérés. A szimuláció eredménye:

```
sal > (display-curr-states)
State 1
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = false
----------------------------
State 2
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
```

```
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = true
-----------------------------
```

7. eset: felhasználónév és ConditionGroup egy paramétere alapján eldöntendő kérés

A következő szimuláció folyamán nem határoztam meg sem a `myname` paraméter, sem pedig a `transProperties` nevű ConditionGroup első (nulladik) paraméterének értékét. A többi beállított paraméter értéke mellett *mars* felhasználónév és a környezeti paraméter *true* érték esetén engedélyezendő, minden más esetben elutasítandó a kérés. A szimuláció eredménye:

```
sal > (display-curr-states)
State 1
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = false
-----------------------------
State 2
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
```

```
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = true
-----------------------------
State 3
--- System Variables (assignments) ---
myname = sec_master
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = false
-----------------------------
State 4
--- System Variables (assignments) ---
myname = sec_master
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = false
-----------------------------
State 5
--- System Variables (assignments) ---
myname = venus
```

```
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = false
-----------------------------
State 6
--- System Variables (assignments) ---
myname = venus
mygroups[0] = User
mygroups[1] = Visitor
resource = Szef
operation = Access
accountProperties[0] = false
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = false
transProperties[1] = false
transProperties[2] = true
transProperties[3] = false
transProperties[4] = true
decision = false
-----------------------------
```

6.4 Modellellenőrzés

A SAL segítségével lehetőség nyílik LTL[17] kifejezések segítségével történő modellellenőrzésre is. Az LTL formulákat a SAL-kódba közvetlenül beleintegrálva, az azokban megfogalmazott logikai kifejezések a `sal-smc` nevű eszköz segítségével ellenőrizhetők. Az eszköz teljes állapottér-bejárással megállapítja, hogy a megfogalmazott feltétel a teljes állapottérben teljesül-e. Ha a feltétel nem teljesül, úgy ellenpéldaként kilistázza azokat az állapotokat, amelyekre a feltétel nem áll fenn.

Fontos, hogy egy valós helyzetben a felállított logikai kifejezések hűen reprezentálják a szabályrendszerrel szemben megfogalmazott követelményeket.

A felállított SAL modellben, a szimuláció során használt szabályrendszerre vonatkozóan felírtam – a teljesség igénye nélkül - néhány olyan logikai kifejezést, amelyek jellemzőek az adott szabályrendszer megkerülhetetlenségére, és ezek teljesülését ellenőriztem:

1. A *venus* nevű felhasználó végrehajthatja a *Szef* nevű erőforráson az *Access* műveletet, a környezeti paraméterektől függetlenül:

```
szabaly1:
     THEOREM system |-
     G((  myname = venus AND
          resource = Szef AND
          operation = Access)
     => X(decision = TRUE));
```

2. A *mars* nevű felhasználó (csoporttagságai *User* és *Visitor*) a *Szef* nevű erőforráson az *Access* műveletet kizárólag a *transProperties* nevű környezeti paraméter-csoport (igaz, hamis, igaz, hamis, igaz) értékkészlete mellett hajthatja végre.

```
szabaly2:
     THEOREM system |-
     G((  myname = mars AND
          mygroups[0] = User AND
          mygroups[1] = Visitor AND
          resource = Szef AND
          operation = Access AND
          decision = TRUE)
```

[17] Linear Temporal Logic, http://en.wikipedia.org/wiki/Linear_temporal_logic

```
        => G(transProperties[0] = true AND
             transProperties[1] = false AND
             transProperties[2] = true AND
             transProperties[3] = false AND
             transProperties[4] = true));
```

3. A *Weboldal* nevű erőforráson csak a *sec_master* nevű felhasználó végezhet *Execute* műveletet:

```
    szabaly3:
        THEOREM system |-
        G((  decision = TRUE AND
             resource = Weboldal)
        => G(myname = sec_master));
```

4. Nem teljesülő kifejezés: a *Weboldal* nevű erőforráson a *mars* nevű felhasználó nem végezhet *Read* műveletet:

```
    kivetel1:
        THEOREM system |-
        G((  myname = mars AND
             resource = Weboldal AND
             operation = Read)
        => G(decision = FALSE));
```

A modellellenőrzés eredménye:

```
Summary:
The assertion 'szabaly1' located at [Context: tam,
line(109), column(2)] is valid.
The assertion 'szabaly2' located at [Context: tam,
line(110), column(2)] is valid.
The assertion 'szabaly3' located at [Context: tam,
line(111), column(2)] is valid.
The assertion 'kivetel1' located at [Context: tam,
line(112), column(2)] is invalid.
```

A 4. kifejezéshez mutatott ellenpélda:

```
Step 2:
--- System Variables (assignments) ---
myname = mars
mygroups[0] = User
mygroups[1] = Visitor
resource = Weboldal
operation = Read
accountProperties[0] = false
```

```
accountProperties[1] = false
accountProperties[2] = false
accountProperties[3] = false
accountProperties[4] = false
transProperties[0] = true
transProperties[1] = false
transProperties[2] = false
transProperties[3] = false
transProperties[4] = false
decision = true
```

Vagyis ezen környezeti paraméter-értékek mellett a *mars* nevű felhasználó elvégezheti a *Read* műveletet a *Weboldal* nevű erőforráson.

6.5 Az eredmények értékelése

A modellellenőrzés segítségével tehát egy meglévő szabályrendszeren formális módszerekkel tudjuk vizsgálni biztonsági kényszerek teljesülését, illetve tudunk mutatni olyan bemeneti paramétereket, amelyek biztonsági kockázatot jelenthetnek a szabályrendszer alkalmazása során. Mindezt még azelőtt megtehetjük, és eliminálhatjuk az esetleges kimutatott kockázati tényezőket, hogy a szabályrendszert alkalmaztuk volna, és ezzel veszélynek tettük volna ki a védett erőforrásokat. A felállított kritériumrendszer újrahasználható, vagyis ezen ellenőrzést érdemes minden olyan esetben elvégezni, amikor a szabályrendszer módosításra kerül, hogy meggyőződhessünk arról, hogy a módosítások nem hoznak-e be biztonsági szempontból sebezhetőségi pontot a rendszerbe.

7 Fejlesztési lehetőségek

A DynRBAC modell, illetve az azt megvalósító AuthEngine szimulációja jelenleg az egyedi felhasználóra és csoportra vonatkozó *ACL*-ekre, valamint a kontextusfüggő attribútumok készletét reprezentáló *ConditionGroup*-okra terjed ki. Bár ezek lefedik a modell működését, az azt megvalósító implementáció, mint Tivoli Access Manager bővítmény fogalomrendszerének további két elemét, a *Protected Object Policy*-kat és az *AuthRule*-kat nem reprezentálja a jelenlegi SAL modell. Ezen két fogalom leképezése, megvalósítása a SAL modellben egy későbbi fejlesztés tárgyát képezheti.

Ugyanakkor az egyre komplexebb jogosultsági szabályok leírásának és betartatásának igénye miatt a hozzáférés-vezérlési módszerek köre napról-napra bővül, újabb és újabb megoldások látnak napvilágot. Egy részük kutatási fázisban maradnak, más részük széles körben használatossá válnak. Ezen munka kitekintést ad az aktuális új hozzáférés vezérlési irányzatokról is, azonban kétségtelen, hogy megírása idején is születnek új, az eddigieknél univerzálisabb, jobb leíró erővel bíró megoldások. Ezek jogosultsági modelljének felállítása és vizsgálata további, e témában folytatott kutatásra ad lehetőséget.

8 Összefoglalás

A diplomaterv során áttekintettem és bemutattam a hozzáférés-vezérlés napjainkban elterjedt koncepcióit, úgymint a Discretionary, Mandatory és Role-based Access Control módszereket, majd kitekintés jelleggel ismertettem a jelenleg még kutatási fázisban lévő, vagy kevésbé széles körben elterjedt megoldásokat is.

Ezt követően előbb szöveges, majd formális módon ismertettem ezen módszerek matematikai modelljét: a jogosultsági döntést meghatározó főbb entitásokat ill. azok egymással való kapcsolatát, valamint az egyes megoldások esetén a jogosultság kiértékelés alapelveit.

Ezután bemutattam a Role-based Access Control módszerét megvalósító egyik konkrét implementációt, a Tivoli Access Manager-t, ismertettem annak előnyeit és a Role-based Access Control modelljének leíró erejéből fakadó hátrányait. Ezen problémákra adott megoldásként áttekintettem a Role-based Access Control-ra épülő, kontextusfüggő paraméterek figyelembe vételének képességével felruházott DynRBAC lehetőségeit, szintén szöveges és formális módon ismertettem annak matematikai modelljét, a jogosultsági döntést meghatározó főbb entitásokat és azok kapcsolatát.

A gyakorlati munka során a DynRBAC modelljét implementáló AuthEngine nevű, Tivoli Access Manager alapú bővítmény entitásait, illetve jogosultság kiértékelő motorjának működését képeztem le SAL modellre. Szimulációt végeztem a modell működésének vizsgálatára, majd modellellenőrzéssel biztonsági kényszerek teljesülését vizsgáltam az eredeti környezetből felolvasott szabályrendszeren.

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani konzulensemnek és kollégámnak, Bősze Tibornak, aki szakmai tudásával, útmutatásával és ötleteivel támogatott ezen diploma elkészítésében; nélküle ez a munka nem készülhetett volna el.

Köszönettel tartozom ezenkívül dr. Pataricza Andrásnak a Budapesti Műszaki Egyetem, valamint Kocsis Zsoltnak az IBM Magyarország részéről, akik tapasztalatukkal és előremutató javaslataikkal mindig segítőkészen támogattak a diplomaterv megvalósítása során.

Végül, de nem utolsósorban köszönetet mondok szüleimnek tanulmányaim során nyújtott szerető támogatásukért és türelmükért.

Ábrajegyzék

Hivatkozások

[1] Wikipedia, the free encyclopedia: Access control; 2008 május
http://en.wikipedia.org/wiki/Access_control

[2] Bősze Tibor - Nagyteljesítményű dinamikus authorizációs szolgáltatás missziókritikus nagyvállalati környezetekhez; 2006
http://lithium.homedns.org/~shanq/dynrbac.html

[3] David F. Ferraiolo, D. Richard Kuhn, Ramaswamy Chandramouli: Access Control Policy, Models, and Mechanisms — Concepts and Examples; 2003
http://www.artechhouse.com/GetBLOB.asp?Name=Ferraiolo_CH2.pdf

[4] Draft: Role Based Access Control Implementation Standard; 2008 május
http://csrc.nist.gov/rbac

[5] Proposed NIST Standard for Role-Based Access Control; 2001 augusztus
http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf

[6] Richard Fernandez: Enterprise Dynamic Access Control Version 2; 2006 január
http://csrc.nist.gov/groups/SNS/rbac/documents/standards/EDACv2overview.pdf

[7] Cisco Documentation: Configuring Context-Based Access Control; 2006
http://www.cisco.com/univercd/cc/td/doc/product/software/ios122/122cgcr/fsecur_c/ftrafwl/scfcbac.pdf

[8] RFC 2575: View-based Access Control Model for the Simple Network Management Protocol; 1999
http://www.faqs.org/rfcs/rfc2575.html

[9] Informatikai Tárcaközi Bizottság: A biztonsági osztályok kialakításának alapelvei; 1996
http://www.itb.hu/ajanlasok/a12/html/a12_4.htm

[10] David Bell: Looking Back at the Bell-La Padula Model; 2005 december
http://www.acsac.org/2005/papers/Bell.pdf

[11] Dénes Tamás – Gondolatok az Internet biztonságáról; 2006 augusztus
http://www.pointernet.pds.hu/ujsagok/evilag/2006-ev/08/20070219203142562000000230.html

[12] UNIX permissions; 1996
http://www.acm.uiuc.edu/webmonkeys/html_workshop/unix.html

Függelék

A SAL forráskód

```
%%Authorization Engine működésének szimulációja

tam: CONTEXT =
BEGIN
  N : NATURAL = 1;
  Index : TYPE = [0 .. N];

  %%Felolvasott User, Group, Protected Object és
  %%Operation listák
  Names: TYPE = {mars, venus, sec_master};
  Groups: TYPE = {Admin, User, Visitor};
  Resources: TYPE = {Szef, Weboldal, Berjegyzek};
  Operations: TYPE = {Read, Write, Execute, Access};
  ConditionGroups: TYPE = ARRAY [0 .. 4] OF BOOLEAN;

  engine : MODULE =
    BEGIN
       GLOBAL myname : Names
       GLOBAL mygroups : ARRAY [0 .. N] OF Groups
       GLOBAL resource : Resources
       GLOBAL operation : Operations

       GLOBAL accountProperties : ConditionGroups
       GLOBAL transProperties : ConditionGroups

       OUTPUT decision : BOOLEAN

       INITIALIZATION
              decision = FALSE;

              %%döntés paraméterei

              %%myname = venus;

              %%mygroups[0] = User;
              %%mygroups[1] = Visitor;

              %%resource = Szef;
              %%operation = Access;

              %%döntési kontextus

              %%accountProperties[0] = false;
              %%accountProperties[1] = false;
```

```
            %%accountProperties[2] = false;
            %%accountProperties[3] = false;
            %%accountProperties[4] = false;

            %%transProperties[0] = true;
            %%transProperties[1] = false;
            %%transProperties[2] = true;
            %%transProperties[3] = false;
            %%transProperties[4] = true;

      TRANSITION
        [
          useracl1:
          (myname = venus AND
           resource = Szef AND
           operation = Access) --> decision'= TRUE

          []
          useracl2:
          (myname = sec_master AND
           resource = Weboldal AND
           operation = Execute) --> decision'= TRUE

          []
         groupacl1:
          (EXISTS (i : Index): mygroups[i] = Admin AND
           resource = Szef AND
           operation = Access) --> decision'= TRUE

         []
          usedconditiongroup_1_authorizedcombination_1:
            (resource = Weboldal AND
             operation = Read AND
             myname = mars AND
             accountProperties[0] = true AND
             accountProperties[1] = false AND
             accountProperties[2] = false AND
             accountProperties[3] = false AND
             accountProperties[4] = false)
             --> decision' = TRUE

          []
          usedconditiongroup_1_authorizedcombination_2:
            (resource = Weboldal AND
             operation = Read AND
             myname = mars AND
             accountProperties[0] = false AND
             accountProperties[1] = false AND
             accountProperties[2] = false AND
             accountProperties[3] = false AND
             accountProperties[4] = false)
```

```
            --> decision' = TRUE

        []
        usedconditiongroup_2_authorizedcombination_1:
          (resource = Szef AND
           operation = Access AND
           myname = mars AND
           transProperties[0] = true AND
           transProperties[1] = false AND
           transProperties[2] = true AND
           transProperties[3] = false AND
           transProperties[4] = true)
           --> decision' = TRUE

        []
        ELSE --> decision'= FALSE

     ]
  END;

system: MODULE =
   engine;

szabaly1: THEOREM system |-
     G((  myname = venus AND
          resource = Szef AND
          operation = Access)
     => X(decision = TRUE));

szabaly2: THEOREM system |-
     G((  myname = mars AND
          mygroups[0] = User AND
          mygroups[1] = Visitor AND
          resource = Szef AND
          operation = Access AND
          decision = TRUE)
     => G(transProperties[0] = true AND
          transProperties[1] = false AND
          transProperties[2] = true AND
          transProperties[3] = false AND
          transProperties[4] = true));

szabaly3: THEOREM system |-
     G((  decision = TRUE AND
          resource = Weboldal AND
          operation = Execute)
     => G(myname = sec_master));

kivetel1: THEOREM system |-
     G((  myname = mars AND
          resource = Weboldal AND
```

```
            operation = Read)
       => G(decision = FALSE));
END
```