Nemzeti Információs Infrastruktúra Fejlesztési Iroda

Azonosítási és jogosultság kezelési rendszer terve

Készítette:
Sándor István
Szalai Ferenc

Készült az Magyar Informatikai Erőforráshalózat (grid közmű) alapjai (MEGA) NKFP OM-00263/2004 projekt keretén belül

NIIF Iroda, Budapest
2006. június

Tartalomjegyzék

1. fejezet

Bevezető

A grid rendszerekben használatos azonosítási és jogosultság kezelési technikák vizsgálata során számos biztonsági és tervezési problémát tártunk fel. Ennek eredményeként úgy döntöttünk, hogy a grid közmű alapját képező Grid Underground (GUG) keretrendszert egy alapvetően új alapokra fektetett azonosítási és jogosultság kezelési alrendszerrel egészítjük ki. Ebben a dokumentumban összefoglaljuk az általunk javasolt azonosítási és jogosultság kezelési keretrendszer architektúráját és működési elvét.

2. fejezet

Rendszerterv

Egy rövid áttekintés után részletesen ismertetjük a GUG biztonsági rendszer elvi működését és a választott megoldásokat. A GUG-hoz tervezett biztonsági szolgáltatás igyekszik kiküszöbölni az az eddig haszált gird biztonsági rendszerek hátrányait, emiatt több ponton is eltér azoktól.

2.1. A feladat

A biztonsági szolgáltatás tehát a GUG rendszerhez fog illeszkedni. A többi szolgáltatással a GUG keretrendszer által biztosított HTTPS/SOAP kapcsolaton keresztül kell kommunikálnia. Pontokba szedve a biztonsági szolgáltatástól a következőket várjuk el:

- A szolgáltatás biztosítsa a felhasználók, szolgáltatások és jobok azonosítását.
- Legyen lehetőség a grid minden erőforrásához hozzáférési jogosultságok hozzárendelésére.
- Az adatbázis legyen elosztott, egyes részei külön-külön adminisztrálhatók legyenek.
- A jogosultságok legyenek átruházhatók.
- Az adminisztrációs lehetőségek lehetőleg egyszerűek és áttekinthetők legyenek.

2.2. Alapgondolatok

Első lépésként felmértök, hogy a szolgáltatás alapú gridben milyen folyamatok zajlanak. Meghatároztuk a szereplőket és erőforrásokat, valamint hogy ezek között milyen kapcsolatokat kell nyilvántartani.

2.2.1. Szereplők

Alapvető szereplők a szolgáltatások. A szolgáltatások erőforrásokat kezelnek, és bizonyos entitások kérésekkel fordulhatnak hozzájuk, aminek hatására a szolgáltatások az erőforrásokon műveleteket hajtanak végre. Egy-egy ilyen műveletkérés jogosságát a biztonsági szolgáltatás segítségével kell tudni elbírálni. A jogosultságkezelés alapegységeinek az entitásokat, erőforrásokat és műveleteket választotuk. Ezek a szereplők szerepelnek a szolgáltatás kérésekben.

Entitások

A gridben három aktív résztvevőt különböztethetünk meg: a *felhasználókat*, a *jobokat* és a *szolgáltatásokat*. Közös tulajdonságuk, hogy aktívan kommunikálnak a szolgáltatásokkal, és a szolgáltatásokon keresztül szeretnének műveleteket végezni az erőforrásokon. A továbbiakban entitás alatt ezt a három szereplőt értjük.

Erőforrások

Erőforrás bármi olyasmi lehet a gridben, amit szolgáltatások kezelnek. A legjellegzetesebb erőforrások a fájlok, de például az ütemezők is tekinthetők erőforrásoknak.

Műveletek

Az erőforrásokhoz műveletek köthetők, például a fájlokat lehet írni és olvasni. A műveletek elvégzését a szolgáltatásoktól lehet kérni.

2.2.2. A jogosultság kérdések általános alakja

A következő lépés a jogosultság kérdések (művelet kérések) alakjának formalizálása volt. Amikor egy entitás egy adott szolgáltatáshoz fordul valamilyen kérelemmel, a következő formájú jogosultsági kérdés vetődik fel:

Végezhet-e az A entitás B műveletet a C erőforráson?

Az elbírálandó kérdések tehát felfoghatók entitásokból, műveletekből és erőforrásokból alkotott hármasokként. Néhány példa:

`<entity_ID_1, read, /grid/niif/home/user12>`: az entity_ID_1 azonosítóval rendelkező entitás olvashatja-e a /grid/niif/home/user12 könyvtárat?

`<entity_ID_2, submit, job_controller_ID_1>`: az entity_ID_2 azonosítóval rendelkező entitás indíthat-e (submitolhat-e) jobot a job_controller_ID_1 azonosítójú Job Controller szolgáltatásnál?

Amint később a 2.4. szakaszban látni fogjuk, a helyzet nem ilyen egyszerű, nem minden jogosultsági kérés fogalmazható meg ilyen egyértelműen. Mindazonáltal, a további megfontolások erre az alapgondolatra épülnek.

2.3. Azonosítás

Az entitásokat és erőforrásokat azonosítókkal kell ellátni. Célszerű volna, ha az aktív szereplőket azonosítóik alapján meg lehetne különböztetni. Emiatt olyan azonosítókat választotunk, amelyek jól megkülönböztethetőek, és viszonylag kézenfekvők.

2.3.1. Entitás nevek

Az azonosításhoz hagyományos X.509 tanúsítványokat használunk, az azonosítók a tanúsítványok DN-jének *emailAddress* mezőjében vannak elhelyezve. Azért esett a választás az emailAddress mezőre, mert a felhasználók azonosítása ezt kívánja meg, a többi entitás szempontjából pedig közömbös, hogy melyik mezőt használjuk. Előny, hogy így a felhasználók bármely olyan tanúsítvánnyal regisztrálhatják majd magukat a rendszerben, amelynek az emailAddress mezője ki van töltve.

Feltételezzük, hogy a felhasználók maguk gondoskodnak saját tanúsítványaik beszerzéséről. A grid szolgáltatásai több kibocsátó CA tanúsítványait is elfogadhatják, ez átjárhatóvá teszi a rendszert más szolgáltatók felé.

A jobok saját tanúsítványokat kapnak, melyeket vagy egy központi grid CA szolgáltatás, vagy a virtuális organizációk állítanak ki, a felhasználó közreműködése nélkül. A jobok jogosultságait saját azonosítójuk szerint tartjuk nyilván, a jobok nem megszemélyesítés útján kapják jogosultságaikat.

A szolgáltatások tanúsítványait szintén a grid CA, vagy az őket üzemeltető virtuális szervezet állítja ki.

Az entitások nevei alapvetően az azokat azonosító X509 tanúsítványok subject DN-jeiből származtathatóak. Az általános elemeket nem terheli megkötés azonban ay utolsó commonName (CN) attribútumra vonatkozóan alább javaslatokat teszünk.

Felhasználók

A felhasználókat e-mail címmel azonosíthatjuk. Egy felhasználó azonosítója tehát például így nézhet ki: `foo@bar.hu`. Az e-mail címek előnye, hogy egyediek, és ennek biztosítására nem kell külön mechanizmust bevezetni. Másrészt az e-mail címek könnyen megjegyezhetőek, közvetlenül kötődnek a felhasználóhoz, és felhasználhatók például üzenetek küldésére.

Jobok

A jobok indításkor UUID-t (Universally Unique IDentifier) kapnak, ami egy 32 bájtos, szabványban rögzített formájú azonosító. Előnye, hogy automatikusan előállítható. Ezzel a választással a jobok azonosítói nem hozhatók közvetlen kapcsolatba a felhasználókkal.

Szolgáltatások

A szolgáltatásokat URL-ekkel azonosítjuk. Az URL első része a szolgáltatást futtató gép neve, a második része pedig a szolgáltatás megnevezése. Szolgáltatás megnevezése lehet például:
`server1.niif.hu/Scheduler`

2.3.2. Erőforrás nevek

Az erőforrás nevek tetszőleges karaktersorozatok lehetnek ezeket minden esetben az a szolgáltatás értelmezi és használja, aki az adott erőforrást reprezentálja a rendszerben. Természetesen vannak esetek, például az állományok, akiknek vannak természetes azonosítói.

Állományok

A fájlokat a hagyományosnak mondható *logikai állománynevekkel* (Logical File Name – LFN) azonosíthatjuk. Ezek „/” jelekkel elválasztott útvonalnevek, például: `/grid/niif/home/file1`

2.4. Jogosultsági viszonyok modellezése

A 2.2. szakaszban már szerepelt, hogy a jogosultsági kérdések általában egy-egy <*entitás, művelet, erőforrás*> hármas formájában merülnek fel.

Első közelítésben tehát tekinthetünk egy olyan adatbázist, amely ezeket a hármasokat tárolja.

2.4.1. További megfontolások

Adatbázisunk tehát ezeket a hármasokat tárolja valamilyen formában. Ha az adatbázisban szerepel egy adott hármas az azt jelenti, hogy az adott entitás az adott műveletet az adott erőforráson elvégezheti. Ha az adott <*entitás, művelet, erőforrás*> hármas nem szerepel az adatbázisban, akkor a művelet nem megengedett.

A továbbiakban az <*entitás, művelet, erőforrás*> hármasok első mezőjét *source* (forrás) mezőnek, a másodikat *action* (művelet) mezőnek, a harmadikat *target* (cél) mezőnek fogjuk nevezni.

Adminisztrációs műveletek

Az adatbázis adminisztrálásához is tartoznak műveletek. Három alapvető műveletet definiáltunk: a jogosultságok listázását (*list*), a jogosultság megadását (*grant*) és a jogosultság visszavonást (*revoke*). Ezeknek a műveleteknek a cél-erőforrása maga a jogosultsági adatbázis, illetve annak bejegyzései. Általános esetben tehát azt mondhatjuk, hogy az ezeket a műveleteket engedélyező jogosultság bejegyzések *target* mezőjében egy jogosultság bejegyzésnek kell szerepelni, például:

> `<entity_ID_1, grant ,<entity_ID_2, read, /grid/niif/home> >`: eszerint az entity_ID_1 entitásnak van joga ahhoz, hogy olvasási jogosultságot adjon az entity_ID_2 entitásnak a /grid/niif/home erőforrásra

Jogosultság átruházás

Szükség van arra is, hogy a felhasználók jogosultságokat adhassanak át a jobjaiknak, a szolgáltatásoknak és esetleg más felhasználóknak is. Emiatt bizonyos jogosultság bejegyzésekhez nekik is szükségük van grant és revoke jogokra.

Kezdetben, az adatbázis inicializálásakor a *root* felhasználó rendelkezik minden jogosultsággal, ő minden erőforrásra rendelkezik grant és revoke jogokkal is. A felhasználók a root-tól kapják saját jogosultságaikat. Ahhoz, hogy a felhasználók is adhassanak át jogosultságokat, például a joboknak, a root-nak meg kell adnia számukra a grant jogot bizonyos erőforrások felett. Szükség van tehát a grant jogosultság átadására is, vagyis a *grant-grant* műveletre, ennek teljes formája a következő:

> `<root, grant, <entity_ID_1, grant ,<entity_ID_2, read, /grid/niif-/home> > >`: eszerint a root entitásnak joga van ahhoz, hogy az entity_ID_1 entitásnak jogot

adjon arra, hogy az olvasási jogosultságot adjon az entity_ID_2 entitásnak a /grid/niif/home erőforrásra

A rekurzív jogosultságok feloldása

Az adminisztrációs műveletek engedélyezéséhez tehát olyan jogosultság bejegyzésekre lenne szükség, amelyek target mezőjükben maguk is jogosultság bejegyzéseket tartalmaznak. A jogosultság átruházás engedélyezéséhez pedig kétszeresen beágyazott jogosultság bejegyzésekre lenne szükség.

Ez a rekurzió gyakorlatilag a végtelenségig folytatható lenne, definiálhatnánk grant-grant-grant jogosultságot is, és így tovább.

Nyilvánvalóan gátat kell szabni ennek a fajta rekurziónak, leginkább azért, mert megnehezíti az adatbázis adminisztrálását. Másrészt nincs is szükség minden esetben ilyen részletes szabályozásra. Az egyik előbbi példát tekintve:

`<entity_ID_1, grant ,<entity_ID_2, read, /grid/niif/home> >`: túlságosan szigorú, ehelyett elegendő lenne:

`<entity_ID_1, grant ,< ... ,read, /grid/niif/home> >`: tehát az entity_ID_1 entitásnak joga van arra, hogy bárkinek írási jogosultságot adjon a /grid/niif/home nevű könyvtárra. Ennél is továbbmenve, elegendő lehet a következő bejegyzés is:

`<entity_ID_1, grant ,<... , ... , /grid/niif/home> >` : vagyis az entity_ID_1 entitásnak joga van arra, hogy bárkinek bármilyen jogosultságot adjon a /grid/niif/home nevű könyvtárra.

2.4.2. A végleges modell

A fenti problémák miatt a következő kompromisszumos megoldást dolgoztuk ki. Mivel egyszerű felépítésű adatbázist szerettünk volna, ezért az előbb említett rekurzív jogosultságokat egyszerűsítetük:

1. Alapesetben a jogosultság bejegyzések csak három szereplőt tartalmaznak: <*source, action, target*>, például:
   (a) `<foo@bar.hu, read, /grid/niif/home>`
   (b) `<foo@bar.hu, grant, /grid/niif/home>`
   (c) `<foo@bar.hu, status, fd0b7216-b7e8-4f3e-8106-4d904ffd5423>`
2. Amelyik jogosultságok ezt igénylik, ott meg lehet adni egy negyedik szereplőt is, ezt *entity_constraint* mezőnek nevezzük. Egyelőre csak a *revoke* jogosultság tartozik ebbe a kategóriába:
   (a) `<root, revoke, /grid/niif/home, foo@bar.hu>`: vagyis a root felhasználó az adott könyvtárra érvényes jogosultságok közül csak a foo@bar.hu névre szólóakat vonhatja vissza.
3. Igény esetén egy további mezőt is hozzá lehet adni a jogosultság bejegyzésekhez, ez az *action_constraint* mező. A revoke jogosultságot például szükség esetén át lehetne alakítani a következőképpen:

(a) `<root, revoke, /grid/niif/home, foo@bar.hu, read>`: vagyis a root felhasználó az adott könyvtárra csak foo@bar.hu-tól vonhat vissza a jogosultságokat, és azok közül is csak a read-et

4. A korlátlan mélységű jogosultság átruházás (a grant jogosultság átruházásával minden jog korlátlanul továbbadhatóvá válna) korlátozása érdekében „grant–grant” műveletet csak a root felhasználó hajthat végre. Tehát egy entitás szerezhet grant jogosultságot egy adott erőforrásra, de más entitásoknak azt nem adhatja tovább.

Csoportok

Az adminisztráció megkönnyítéséhez mindenképpen szükség van csoportokra. A csoportok entitásokat tartalmazhatnak. A csoportok nevei a megkülönböztethetőség kedvéért #-tel kezdődnek és azzal is végződnek. A két # között betűk, számok, aláhúzás és kötőjel szerepelhetnek. Csoportnév lehet például `#jobs#` az összes helyi job jelölésére.

Fontos megjegyezni, hogy *a csoportok nem grid szintű entitások*, csak egy-egy adatbázison (lásd 2.5. szakasz) belül, helyileg érvényesek, és bevezetésükre pusztán a helyi adminisztráció megkönnyítése miatt került sor.

A csoportnevek állhatnak az entitások nevei helyett, például: `<#jobs#, read, /grid/bme-/home>`

Másrészt a csoportok maguk is szerepelhetnek erőforrásként, mivel a csoportokon is lehet műveleteket végezni (entitások hozzáadása és törlése), például: `<foo@bar.hu, addmember, #jobs#>`

Minden adatbázisban van egy speciális csoport, a `#root#` csoport, amelynek egyetlen tagja van, a root felhasználó. A `#root#` csoporthoz nem lehet felhasználókat hozzáadni, és a root felhasználót sem lehet kitörölni onnan.

2.4.3. Erőforrás leírók

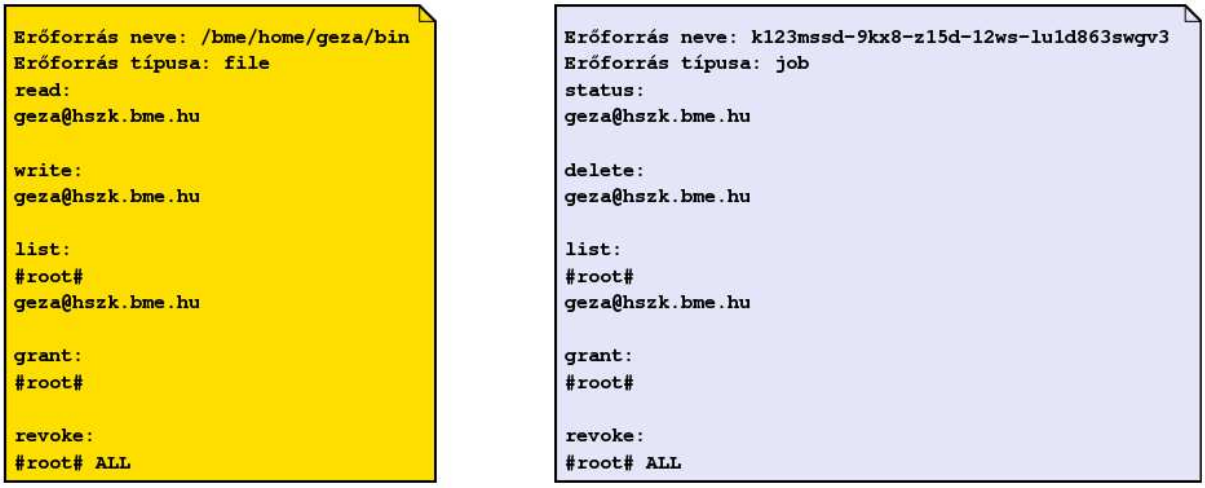
Az adatbázis felépítésének megtervezésekor áttekinthető struktúrára törekedtünk, ezért úgy döntöttünk, hogy a hármas (vagy négyes, ötös) relációkat nem egyenként, hanem rendszerezve tároljuk. A rendezési szempont, hogy az egy erőforrásra vonatkozó jogosultságok együtt legyenek, mint például az ACL-ek esetében.

Az erőforrások szerinti csoportosítás előnye, hogy ez a rendszergazdák által megszokott megjelenítési mód (pl. ls -l). Emellett logikailag a műveletek is az erőforrásokhoz köthetők.

Az erőforrás leírók felépítése

A 2.1. ábrán két erőforrás leíró vázlata látható. Az erőforrás leíróknak van nevük, típusuk, és műveletek vannak definiálva hozzájuk. A leírók a rajtuk definiált műveletekhez bejegyzéseket tartalmaznak. Egy-egy bejegyzés értelmezése: az adott entitás (*source*) az adott műveletet (*action*) az adott erőforráson (*target*) elvégezheti.

Bizonyos műveletekhez (jelenleg csak a *revoke*) két tagból álló bejegyzések tartoznak. A második tag az *entity_constraint* mező. Olyan bejegyzések is alkothatók, amelyek három tagból állnak (*action_constraint*).



2.1. ábra. Erőforrás leíró elvi felépítése

Láthatóan, a fájl típusú erőforráshoz read és write műveletek vannak definiálva. A job típusú erőforráshoz status és delete. Minden erőforráshoz definiálva van továbbá a list, grant és revoke művelet, amelyek a jogosultságok átadását (grant), visszavonását (revoke) és listázását (list) engedélyezik.

A két példa erőforrás leíróban csak a `#root#`-nak van grant és revoke jogosultsága. A revoke alatt látható `#root# ALL` bejegyzés azt jelenti, hogy a `#root#` bárkitől visszavonhat jogosultságot. Egyébként más felhasználók csak azoktól az entitásoktól vonhatnak vissza jogosultságot, amelyeknek ők adtak jogokat. Az erőforrás leírók működésével, típusaival és megvalósításukkal részleteit a megvalósítás során dolgozzuk ki.

Egyéb bejegyzések

Adminisztrációs célból szükség van egyéb bejegyzésekre is, amelyek nem tekinthetők jogosultságoknak. Például a csoportokhoz is tartozik egy-egy erőforrás leíró. Ezeknek a leíróknak van egy *members* „művelete”, amelynek bejegyzései jelölik a csoport tagjait.

Hasonlóan a felhasználó, job és szolgáltatás típusú erőforrás leírókhoz tartozhat *groups* „művelet” is, amelynek bejegyzései azokat a csoportokat tartalmazzák, amelyekben az adott entitás tag.

Ezek megvalósításával és egyéb adminisztrációs bejegyzésekkel szintén magvalósítás során fogunk foglalkozni.

Erőforrás leírók létrehozása

Mindeddig feltételeztük, hogy az adatbázis az erőforrás leírókkal valamilyen módon rendelkezésre áll, és a gridben létező erőforrásokat reprezentálja. Az erőforrásokat, amelyekről nyilvántartást kellene vezetni, azonban különböző szolgáltatások kezelik. A biztonsági szolgáltatásnak a rendszer elosztott volta miatt nincs módja, hogy minden az erőforrásokat érintő változást pontosan nyomon kövessen.

A biztonsági adatbázis állapota tehát sohasem fogja pontosan tükrözni a rendszer állapotát. Hogy az adatbázis mégis úgy működjön, ahogy az elvárható lenne, két szabályt vezetünk be:

1. Az entitásokat explicit módon kell létrehozni (a biztonsági szolgáltatás *create* függvényével)

2. A fájlok automatikusan jönnek létre úgy, hogy az adatbázisban megtalálható leghosszabb prefixumra érvényes jogosultságokat öröklik. Tehát ha egy olyan fájl hozzáférési jogosultságait szeretnénk lekérdezni, ami még nincs az adatbázisban, akkor az új fájl a felette lévő könyvtáraktól örökli a jogosultságokat.

2.5. Elosztott jogosultsági adatbázis

A jogosultsági adatbázis a grid szerkezetéből adódóan nem lehet központosított. Az egyes rész–adatbázisokat egymástól függetlenül adminisztrálják.

Célszerű az adatbázist virtuális szervezetek szerint felosztani, mert így a logikailag összertartozó bejegyzések egy helyre kerülnek. Az általános definíció szerint a virtuális szervezet felhasználók, szolgáltatások és erőforrások közössége, amelyeket közösen adminisztrálnak, én ezekhez hozzávetjük még a jobokat és csoportokat is.

Virtuális szervezetek lehetnek például vállalatok, egyetemek, egyetemi tanszékek, kutatóintézetek, kutatócsoportok stb.

2.5.1. VO szerverek

Az adatbázis tehát virtuális szervezetek szerint (továbbiakban VO) van elosztva a 2.2. ábrán látható módon. Minden VO a globális virtuális fájlrendszer egy-egy részfáját adminisztrálja. Ezen kívül minden VO-ban vannak felhasználók, szolgáltatások, jobok és csoportok. Egy felhasználó, job vagy szolgáltatás akkor tekinthető egy adott VO-ba tartozónak, ha az adott VO erőforrásaira rendelkezik jogosultságokkal. A felhasználók, szolgáltatások és jobok nincsenek VO-hoz kötve, tetszőleges számú VO-ban rendelkezhetnek tagsággal. Ezzel ellentétben minden egyes fájl erőforrást csak egy VO szerver tarthat nyilván.

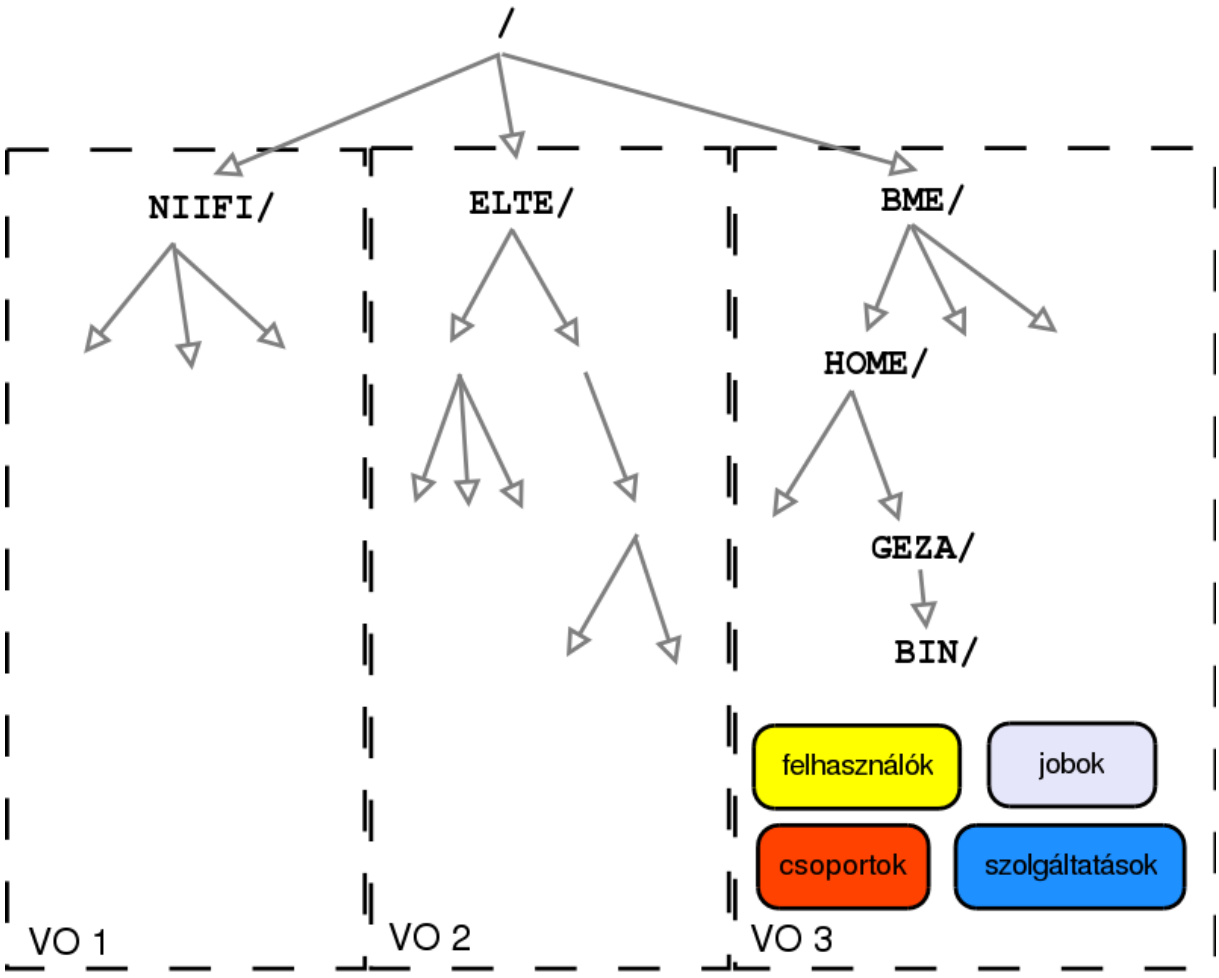
Minden virtuális szervezethez tartozik egy ún. VO szerver, amely az adott VO-hoz tartozó adatbázisrészt tárolja. A VO szerver maga is egy szolgáltatás, amely a GUG keretrendszerre épül. A VO szerver szolgáltatással lehet *grant*, *revoke* és *list* műveleteket végeztetni.

2.5.2. A VO szerver mint CA

A felhasználók saját maguk szerzik be tanúsítványaikat. A jobok és szolgáltatások azonban nem olyan „állandó” szereplők, mint a felhasználók, és nem képesek saját tanúsítványokat beszerezni magukat.

Különösen a jobok indítása gyakori esemény. Mivel a jobok tanúsítványait nem a felhasználó tanúsítványából származtatjuk (nem használunk job proxy tanúsítványokat), ezért a jobok tanúsítványait a VO szerverek írják alá. Ezen kívül a szolgáltatások tanúsítványait is a VO szerverek írják alá. Ezek a tanúsítványok egyben ún. *tagsági igazolványok* is, ezekkel a következő szakasz foglalkozik.

Az ilyen tanúsítványok kibocsátásához a VO szerver is rendelkezik CA funkciókkal. Minden VO szervernek van saját titkos kulcsa és CA tanúsítványa, amelyekről feltételezzük, hogy a VO szerverek képesek titokban tartani azokat. Úgy is mondhatnánk, hogy a szolgáltatások megbíznak a VO szerverek által kibocsátott tanúsítványokban.



2.2. ábra. A névtér felosztása VO szerverekre

2.6. Tagsági igazolványok

A 2.4. szakaszban ismertetett (hármas, négyes vagy ötös) jogosultság bejegyzéseket a VO szerverek tárolják. Mielőtt egy szolgáltatás engedélyezné a hozzáférést egy adott erőforráshoz, meg kell tudnia, hogy szerepel-e a megfelelő jogosultság bejegyzés az adatbázisban. A lekérdezést a szolgáltatás egy API-n (Application Programming Interface) keresztül végzi. Az adatbázis elosztott, a rendszerben tetszőleges számú VO szerver szerepelhet, így az összes szerver lekérdezése nem jó megoldás. Meg kell találni azokat a szervereket, ahol a hozzáférési kérelemmel jelentkező entitás nyilván van tartva.

Tagsági igazolványok

Az ötlet az volt, hogy amikor egy a felhasználót, jobot vagy szolgáltatást felvesznek egy virtuális szervezetbe, a VO szerver kiállít számára egy ún. tagsági igazolványt. A tagsági igazolvány egy hagyományos X.509 tanúsítvány, melynek *Issuer/emailAddress* mezőjében a VO szerver címe szerepel. A tagsági igazolvány *Subject*-je a felhasználó, job vagy szolgáltatás.

A tagsági igazolvány azt igazolja, hogy az adott entitás az adott VO szerveren nyilván van tartva. Hitelessége a VO szerverek nyilvános kulcsainak birtokában ellenőrizhető.

A tagsági igazolványok nagy előnye az attribútum és proxy tanúsítványokkal szemben, hogy érzékeny információt nem tartalmaznak. Nem azonosításra szolgálnak, hanem csak annak eldöntésére, hogy az adott felhasználó esetében a szolgáltatás melyik VO szerverhez forduljon a jogosultság elbírálása érdekében. Felhasználók esetén a tagsági igazolványok csak a felhasználó tanúsítványával együtt érvényesek.

A szolgáltatások és jobok nem képesek saját tanúsítványaik hatékony védelmére. Leginkább a jobok sebezhetők ebből a szempontból. Másrészt jobok indításakor és szolgáltatások telepítésekor nem kívánunk kapcsolatba lépni a CA-val. Ezért ezeknél a tagsági igazolványok egyben az azonosítást is szolgálják.

Az igazolványok felhasználása

Az igazolványokat a biztonsági rendszer által kínált API használja fel a jogosultsági kérdések irányítására (route-olására). A tagsági igazolványokat az X.509 tanúsítványokkal együtt a keretrendszer továbbítja, így azok a felhasználók elől rejtve maradnak, és így továbbítódnak a szolgáltatásokig.

Amikor a szolgáltatások az API valamely függvényét meghívják, a átadják a függvény számára a szolgáltatást kérő entitás tagsági igazolványait. Az API függvény a tagsági igazolványok *Issuer* mezőjéből kiolvassa a VO szerverek címeit, és ezekhez a VO szerverekhez továbbítja a jogosultsági kérdést.

2.7. Proxyzás, Single Sign On

Bár a proxy tanúsítványok nem használhatók biztonságosan jogosultság átruházásra, az egyszeri bejelentkezés (Single Sign On) megvalósítására azonban alkalmasak lehetnek. Erre vannak létező megoldások, például a MyProxy.

Másik lehetőség valamilyen agent-program (amilyen például az ssh-agent) alkalmazása, amely a memóriában tárolná a titkos kulcsokat.

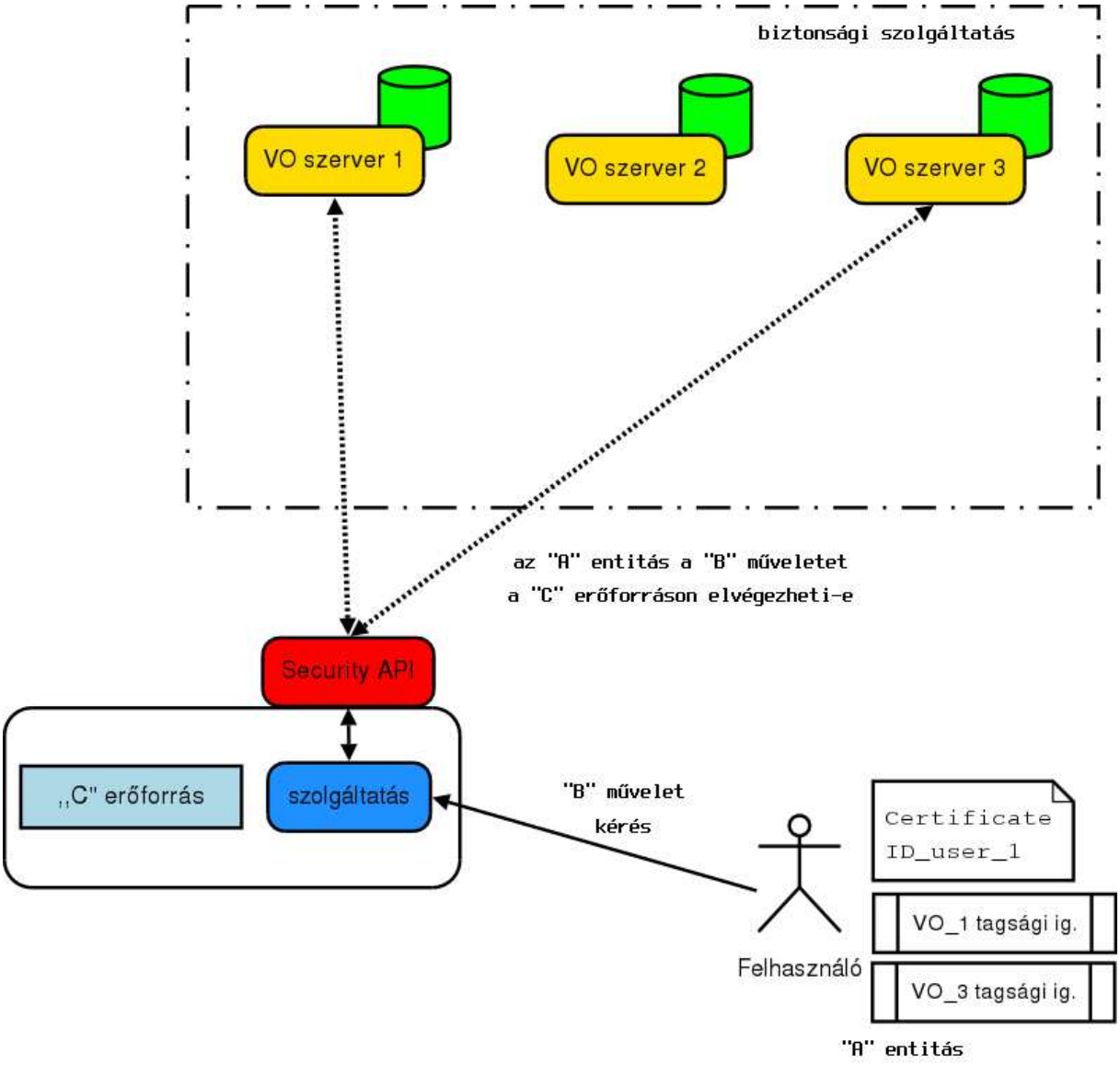
A GUG biztonsági rendszernek jelenleg ez nem része, viszont a későbbiekben mindenképpen szükség lesz rá.

2.8. A rendszer működése

Az alábbiakban egy rövid példán mutatjuk be a rendszer működését. A példában egy felhasználó egy adott fájlt kíván letölteni a Storage-ról.

1. A felhasználó először a Storage szolgáltatáshoz fordul, megadva a fájl logikai fájlnevét, például: `/grid/niif/home/a.txt`
2. A Storage szolgáltatást a felhasználó a GUG kliens oldali interfészen keresztül hívja, például az `ls` műveletet. A kliens oldali interfész gondoskodik róla, hogy a keretrendszer megkapja a felhasználó tanúsítványát és tagsági igazolványait, amelyek egy előre megadott könyvtárban találhatók. A HTTPS kapcsolat kiépülésekor a felhasználónak meg kell adnia a tanúsítványhoz tartozó titkos kulcsot védő jelszót, ezzel a felhasználó azonosította magát.
3. A keretrendszer, melynek függvényeit a kliens oldali interfész is használja, továbbítja a Storage szolgáltatáshoz mind az X.509 tanúsítványt, mind a tagsági igazolványokat.
4. A Storage szolgáltatás ezután a biztonsági API-t hívja, átadva neki a felhasználó tanúsítványát és tagsági igazolványait, valamint a kérésre vonatkozó adatokat, körülbelül ebben a formában: `(tanúsítvány, tagsági ig. lista, 'read','/grid/niif/home/a.txt')`
5. A biztonsági API ellenőrzi, hogy a tagsági igazolványok aláírása érvényes-e, tehát érvényes VO szerver bocsátotta-e ki. Másrészt az Issuer mezőiből meghatározza a szóban forgó VO szerverek címét, és a keretrendszeren keresztül meghívja az adott VO szerver szolgáltatások megfelelő függvényét.
6. A VO szerver megállapítja, hogy szerepel-e `<felhasználó_ID, 'read', '/grid/niif-/home/a.txt'>` hármas az adatbázisban. Ha igen, akkor *True* választ ad, ha nem, akkor *False* választ.
7. A Storage szolgáltatás a válasznak megfelelően folytatja, vagy nem folytatja a fájl előkeresését.

A rendszer működését a 2.3. ábra szemlélteti.



2.3. ábra. Kérés jogosságának ellenőrzése