MISKOLCI EGYETEM
GÉPÉSZMÉRNÖKI ÉS INFORMATIKAI KAR

TUDOMÁNYOS DIÁKKÖRI DOLGOZAT

Címkézett dokumentum-nyilvántartás felhasználóbarát kezelése és alkalmazása

Piller Imre
Mérnök informatikus MSc, II. évfolyam

Konzulensek:

Dr. Fegyverneki Sándor
egyetemi docens, tanszékvezető
Alkalmazott Matematikai Tanszék

Dr. Kovács László
egyetemi docens, tanszékvezető
Általános Informatikai Tanszék

Miskolc, 2012

Tartalomjegyzék

1. fejezet

Bevezetés

A dokumentumok, állományok tárolása döntően hierarchikus formában történik. Ez már a legelső adattárolók megjelenése óta így van, és már annyira természetessé vált, hogy nehéz elképzelni azt, hogy esetleg más megoldás is lehetséges lenne. Az objektumok hierarchiába történő rendezésének gondolata már jóval az első számítógépek megtervezése előtt megjelent. Az informatika ezen, a korábban már jól bevált elven haladt tovább.

A számítógépek segítségével manapság már rengeteg információt tudunk tárolni a különféle rendszerekben. A fájlok tárolási koncepciója a fájlrendszerekben közismert, de mégis érdemes lehet pár szóban összefoglalni, hogy fény derüljön a módszer előnyeire és hátrányaira, illetve egyszerűbb legyen majd az összevetése az újszerű megközelítéssel.

A tárolási mód építőeleme a jegyzék. Ebben állományok és aljegyzékek is lehetnek, amiből így kiépíthető egy fastruktúra, melyben mindennek egy konkrét jól meghatározott helye van. Minden jegyzéknek és állománynak kell, hogy legyen neve. A struktúrában az elemeket az ezek segítségével megadott útvonaluk egyértelműen kell, hogy azonosítsa őket.

A hierarchikus tárolásnak rengeteg előnye van. A felhasználó szempontjából talán a legfontosabb, hogy egy megszokott struktúrát használhat, könnyebben át lehet látni a rendszer, és keresni benne. Megtartja azt a gyakorlati kötöttséget, hogy minden egyszerre csak egy helyen lehet, ha egy állományt egy jegyzékben elhelyezünk, akkor értelemszerűen azt ugyanott fogjuk majd keresni is, általában a hierarchiában lefelé haladva az egyedi útvonal mentén.

A fastruktúra a tárolás szempontjából is előnyös. A jegyzék tulajdonképpen csak egy különleges fájltípus, amely más fájloknak a listáját tartalmazza. Ezek tartalma viszonylag egyszerűen kilistázható, rekurzív eljárásokkal így bejárható az egész struktúra.

Az eddigiekben nem esett szó a hierarchikus fájlrendszerekben használt néhány különleges megoldásról.

Vannak helyzetek, amikor szükség lehet például arra, hogy egy állományt több helyről is elérjünk. Ezekre a hierarchikus fájlrendszerek biztosítanak néhány megoldási lehetőséget. A legegyszerűbb ezek közül, hogy az állományt átmásoljuk, viszont ez indokolatlan redundanciához vezet. Létrehozhatunk linkeket is. Ez lehet csak egy utalás az állományra (*soft link*), vagy pedig ugyanazon állományra történő hivatkozás több különböző helyről (*hard link*). Az előbbit kezelhetjük, mint egy különleges fájltípusnak, az utóbbi viszont már nehezebben illeszthető az eddigi fastruktúrába, mivel az elérési útvonal egyediségére vonatkozó feltevésnek mond ellent. Ezek mindenesetre már beépültek a napjainkban használatos rendszerekbe, így létjogosultságukat nem lehet megkérdőjelezni. Használatuk különös odafigyelést igényel; el kell kerülni a körkörös

hivatkozásokat, illetve azon rekurzív algoritmusoknál is körültekintően kell eljárni, ahol ezek nem várt mellékhatásokat okozhatnak.

A hivatkozások megjelenése azt jelzi, hogy a hierarchikus tárolás kötöttségeit alkalmanként lazítani kell. Az egyik vizsgálandó kérdés az, hogy vajon elegendő néhány speciális elem bevezetése, vagy célszerűbb lehet-e egy más megközelítést alapul venni.

1.1. *A hierarchikus tárolás problémái*

Az elemzésünket először a felhasználó szempontjából végezzük el. Egyik ilyen probléma a kategorizálás. Sokszor a kategóriába sorolás nem egyértelmű.

Erre tipikus példa a zeneszámok tárolása. Az egyik lehetőség, hogy előadónként hozunk létre jegyzékeket, majd ezeken belül műfajonként, vagy albumonként. Elképzelhető az is, hogy a csoportosításnál a műfaj az elsődleges, az előadókat és albumokat majd csak ezen belül szeretnénk szétválogatni. Bármelyiket is választjuk könnyen előfordulhat, hogy nem tudjuk az állományainkat megfelelő helyre tenni, mert például egy albumhoz több előadó, vagy több műfaj is tartozik (esetleg mindkettő). Az egyik gond tehát, hogy ott kellene alá-fölé rendeltséget megállapítani, ahol az nem lehetséges.

A felhasználóknak szükségük van a gyors keresésre. A klasszikus fájlrendszerek ebből a szempontból korlátozottak, ezt közvetlenül nem támogatják. Vannak természetesen módszerek, de azok rendszerint a fastruktúra bejárásával keresnek, amely elég költséges. A keresési szempontok körébe általában csak a név, típus illetve módosítás dátuma szokott beletartozni.

Az állományok megtalálásához a jegyzék-, és fájlneveknek segítséget kell, hogy nyújtaniuk, de az már a felhasználón múlik, hogy milyen mértékben. Az előző bekezdésben felvetett kategorizálási problémán kívül még további nehezítő tényezők játszhatnak közre.

Az egyik ilyen, ha nagyobb mennyiségű adat átvétele esetén (például archiválás, biztonsági mentés) nem volt lehetőség minden fájlt a megfelelő helyre rakni. Az egész egy jegyzékbe került, amely inkább utal a származási helyére, mint a tartalmára.

Problémát okozhat, ha a dokumentumok verziókezelését a fájlrendszer adta lehetőségekkel sikerült megvalósítani, és a régebbi verziókra sem csak a munka előrehaladásának vizsgálata, vagy biztonsági másolat szempontjából van szükség.

Az állományaink jelentős részének nem mi választunk nevet. Az Internetről származó fájlokat, fényképezők vagy más eszköz által generált azonosító jellegű neveket persze átírhatjuk, de hogy ez minden esetben megtörténjen az szigorú következetességet igényel, és az gyakran felesleges erőfeszítésnek bizonyul. A probléma itt igazán csak akkor jelentkezik, amikor az állomány helyét illetően minimális információ áll rendelkezésünkre. A keresés ilyenkor típus, dátum, esetleg fájlméret alapján lehetséges, ami nem mindig vezet eredményre.

Az Internet más nézőpontból is jelentősen befolyásolta a felhasználói szokásokat. A felhő alapú technológiákkal szemben még mindig megvan a bizalmatlanság, viszont ez inkább a biztonsági mentésekre vonatkoztatva kulcskérdés. Hajlamosak lehetünk azt hinni, hogy nálunk nagyobb biztonságban van az adott állomány, viszont arról már kevesebbet esik szó, hogy mennyi idő alatt tudjuk azokat előkeresni a saját tárolónkról.

A fájlrendszer vagy dokumentumnyilvántartás megtervezése és kivitelezése szempontjából a hierarchikus megoldással igazából nincs semmi gond. Technikai szempontból persze nem triviális a kérdés, viszont a jegyzék fogalmából kiindulva szép, letisztult szerkezetek valósíthatók meg.

1.2. *Címkézés alapú megközelítés*

A dolgozatban egy újszerű, kifejezetten csak címkézésen alapuló dokumentum-nyilvántartási módszerről lesz szó. Címkézést elég sok helyen használnak, ezért pár szóban érdemes összefoglalni, hogy a bemutatásra kerülő koncepció hova is tartozna, és miben jelentene újat.

A címkézést az alkalmazásokban mint egy kiegészítő funkciót szokták használni, jelen esetben viszont ez a rendszerezés alapja. Az elsődleges cél tehát nem egy új absztrakciós réteg kialakítása a meglévő hierarchikus rendszerek felett, hanem annak vizsgálata, hogy egy ilyen rendszer alkalmas lehet-e azok kiváltására.

A címke szerepe itt a jegyzékéhez hasonló. A címkézés automatizálására csak annyiban lehet szükség amennyiben a jegyzékek esetében is. A program tehet javaslatokat erre vonatkozóan, de nem feltételezi, hogy a felhasználó helyett döntést tud hozni a dokumentum elhelyezését illetően.

A formális koncepciók elemzésével felderíthető a hierarchikus- és a címkézett nyilvántartás közötti kapcsolat. A fogalmi hálók adják majd azt a struktúrát, mely szemléltethetővé teszi a dokumentumok és címkéik viszonyát.

A felhasználók általában nem szeretnek dokumentumaikhoz címkéket rendelni. Az egyik ok a felhasználói felületek kialakításában keresendő. Bemutatásra kerül majd egy új felépítés, illetve a használati módja, amely remélhetőleg megoldja az ilyen jellegű problémákat.

A felhasználói felület első változata egy webalkalmazás volt. Ez még a dokumentumok adatainak tárolásához egy elterjedt dokumentum-orientált adatbázist használt. Később szükségesnek látszott egy egyszerűbb, speciálisan a címkék és dokumentumok adatainak tárolására szolgáló adatbázismotor elkészítésébe kezdeni.

A címkézés egyik fő felhasználási területe a fájlok kezelése lehet, ezért egy fájlkezelő program tervei és alapvető funkcióknak az implementációja is szerepel a dolgozatban.

Az említett programok, és a forráskódjaik a

```
users.iit.uni-miskolc.hu/~piller/Cimke
```

címen érhetők el.

A bemutatott kutató munka a TÁMOP-4.2.1. B-10/1/KONV-2010-0001 jelű projekt részeként az Európai Unió támogatásával, az Európai Szociális Alap finanszírozásával valósul meg.

2. fejezet

Címkézést használó alkalmazások

2.1. *Fájlkezelő programok*

A kényelmesen használható felhasználói felület kialakítása döntő fontosságú. Érdemes megvizsgálni először milyen elterjedt kereső- és böngészőfelületek vannak, illetve azt, hogy minek köszönhetik a népszerűségüket.

A kezdetben használt rendszerek parancssoron keresztül tették lehetővé az állományok visszakeresését. A két fő művelet a jegyzék tartalmának listázása (`ls` vagy `dir`) illetve a jegyzék váltása volt (`cd`).

Ez után olyan programok jelentek meg, amelyek még szintén karakteres felületet használtak, de már a jegyzék váltását követően azok tartalmát automatikusan listázták, illetve volt egy aktuálisan kijelölt elem a listában, amin a különféle műveleteket már a fájlnév direkt megadása nélkül el lehetett végezni. A dupla paneles kialakítás bizonyult igazán szerencsésnek ezek közül, mivel a gyakran alkalmazott fájlműveletek között sok olyan van, amelyiknél jó, ha a forrás- és a céljegyzék egymás mellett látható.

A grafikus megjelenítéssel egy jegyzék tartalma külön ablakba kerülhetett. A fájlokat már ikonok formájában jelenítették meg a fájlkezelő programok, ami így már jobban utalhatott a fájltípusukra is. Egyre szemléletesebbé válhatott az, hogy mi hol helyezkedik el a fájlrendszerben. Az áthelyezés művelete a felhasználó szemszögéből már valóban úgy jelenik meg, hogy a fájl az egyik helyről átkerül egy másikra. Az aktuális jegyzék teljes elérési útvonala tipikusan az adott alkalmazás ablakának felső részébe, vagy magába a címsorba kerül. Bal oldalt egy könyvjelzősáv vagy fastruktúra nézet szokta még segíteni az egyszerűbb böngészést. Vannak ez alól természetesen kivételek, illetve a felület személyre szabásával át lehet ezt az elrendezést alakítani, viszont vélhetően szoftverergonómiai okokból vált ez hagyományossá.

A címkézés a jelenlegi programokban, mint egy plusz funkció jelent meg, de lényegi változást nem okozott a felépítésükben. Általában egy címkekezelő felületen szerkeszthetjük az állományok címkéit, és egy külön panel szolgál arra, hogy ezek alapján tudjunk majd keresni.

A klasszikus fájlkezelők közismertnek tekinthetők, ezért nem érdemes részletekbemenően bemutatni őket. A címkézést használó alkalmazások közül a jelentősebbek sorravétele több szempontból is szükséges lehet. Egyrészt képet kaphatunk a jelenleg elérhető funkciókról, megvizsgálhatjuk a felhasználási lehetőségeket. Fontos még továbbá elemezni a felépítésüket, az egyes funkciók használatának módjait illetve a belső megvalósításukat is ahol ez lehetséges. A következő részekben ilyen alkalmazások bemutatására kerül sor.

2.1.1. *Elyse*

Az Elyse[1] Egy Windows és Mac OS operációs rendszerek alatt futtatható, szabadon hozzáférhető címkézést alkalmazó fájlkezelő. A jegyzék koncepciót címke csomópontokkal (*tag nodes*) váltja föl. Ezek előre definiált kereséseknek tekinthetők, amellyekkel így gyorsabban tudunk navigálni, mint ha minden alkalommal külön keresési utasításokat adnánk ki. A csomópontok fákba rendezhetők, így hasonló hatás érhető el, mint ha azok jegyzékek lennének (2.1 ábra). A különbség ott jelentkezik, hogy egy fájl több ilyen csomópont által reprezentált csoportba is tartozhat.

A címkék közötti kapcsolatok kezelését is lehetővé teszi, a címkék szintén csoportokba rendezhetők, amivel így az egyes fájlokon lévő címkék számát csökkenteni tudja, és általánosabb keresési műveleteket tesz lehetővé.

A címkézés *drag-and-drop* módon és párbeszédablakon keresztül is elvégezhető benne. Egy automatizált funkciója, hogy a már meglévő jegyzékstruktúrákból felépíthető vele a címkeadatbázis. A konverzió közben figyeli, hogy a fájlok nincsenek-e meg több példányban is, és ha így lenne figyelmezteti a felhasználót.

Az alkalmazás adatbázisa egyetlen fájlban kap helyet. Ennek maximális mérete az operációs rendszertől, és a használt fájlrendszertől függ. A formátuma olyan, hogy valamilyen szinten platformfüggetlen lehessen, tehát Windows és Mac OS rendszerek között átvihessük az állományainkat a hozzájuk tartozó metaadatokkal együtt.

Az állományok tárolásához két módot használ:

- *External*: A fájl a külső fájlrendszerre hivatkozik. A program használata így kiegészítheti a hagyományos módszereket, más alkalmazások is használni tudják.
- *Internal*: A program az adatbázishoz hasonlóan saját állományban tárolja. Ennek az előnye, hogy a hivatkozások nem romolhatnak el, viszont más alkalmazások nehezebben férhetnek hozzájuk.

Címkéket állományoktól függetlenül a *Tags dock* segítségével lehet létrehozni és szerkeszteni. A böngészéshez a *Browsing Tree* panel szolgál, amin fastruktúrában a csomópontokkal adhatók meg a szűrési feltételek. Ezen hat előre definiált csomópont van, méghozzá: *All files*, *All files with tags*, *All files without tags*, *All internal files*, *All external files*, *Recently added files*.

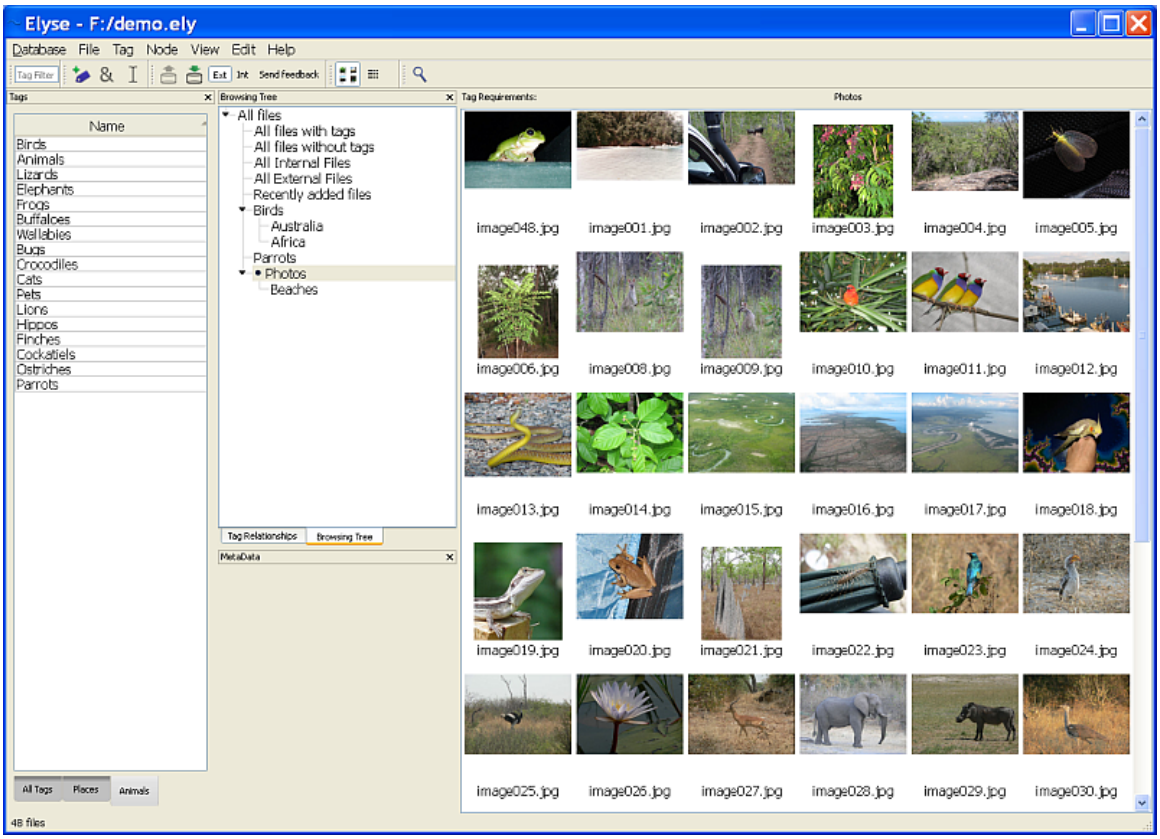
A címkék közötti kapcsolatokhoz a *Tag relationships tab* fül használható. A címkék között szülő-gyermek kapcsolat alakítható ki. A metaadatok szerkesztéséhez szintén külön *Metadata dock* szolgál. A program a fájlok és metaadatok importálásához és exportálásához is ad eszközöket, illetve jelzi ha hiányozna a külső fájl.

2.1.2. *Windows Future Storage*

A *Windows Future Storage*-t Microsoft fejlesztette Windows operációs rendszerek számára, mint magasabb szintű adattárolási alrendszert. Struktúrált és kevésbé struktúrált adatok tárolására készült, az adatokat pedig a háttérben relációs adatbázisok szolgáltatják. 2003-ban mutatták be, viszont a rendszer fejlesztését időközben leállították, így nem kerülhetett sor ilyen formában a bevezetésére.

Egy egységesített típusrendszert használ sémákkal definiálva. Ezzel az alkalmazások egyszerűbben oszthatják meg egymással az adataikat, a kapcsolatok és a lehetséges szűrési módok is közvetlenül rendelkezésre állnak. 2006-ban még fejlesztési fázisban volt, ezért inkább úgy döntöttek, hogy bizonyos részeit más alkalmazásokban, mint például a Microsoft Semantic Engine-ben és az SQL Server-ben használják fel.

[1] [Elyse] http://silkwoodsoftware.com



2.1. ábra. Elyse, tag címkézést alkalmazó fájlkezelő. A címkéket további struktúrába lehet rendezni.

Az NTFS fájlrendszer fölé épült, maga nem egy külön fájlrendszer volt. A limitált keresési lehetőségeket próbálta kompenzálni a metaadatok nyilvántartásával és rendszerezésével. A felhasználó szemszögéből ez fejlettebb, tartalomra történő keresési funkciókat jelentett volna akár természetes nyelvi lekérdezések formájában is. Egyik böngésző ehhez a *StoreSpy*, amely az elemek jellemzői és kapcsolatai alapján keres a struktúrált adatok között.

Ebben is megjelenik a virtuális mappa (*virtual folder*) koncepció. Ezek egyik típusa a keresési mappa (*search folder*), amelyet megnyitva egy keresés fut le a háttérben, és annak az eredményét jeleníti meg a fájlböngésző. A Windows 7 is használ virtuális mappákat, melyek egyik megjelenési formája a könyvtár (*library*).

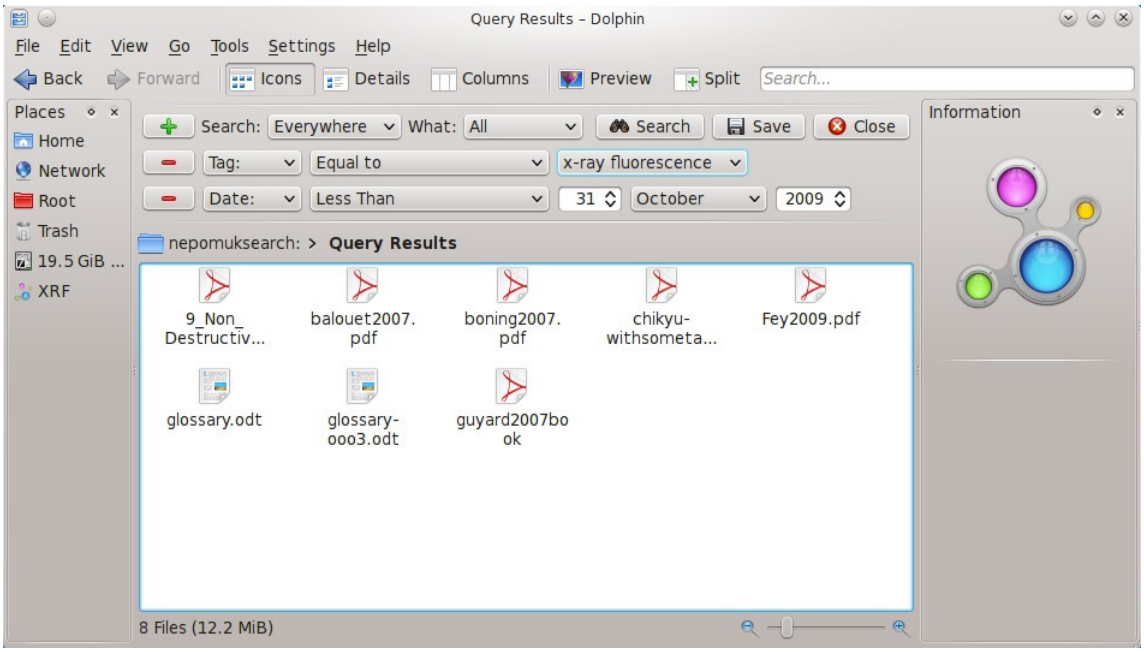
2.1.3. *Nepomuk - The Social Semantic Desktop*

A név a *Networked Environment for Personalized, Ontology-based Management of Unified Knowledge*[2] kifejezésből ered. Az Európai Unió egy nagy méretű kutatási projektjéről van szó, amelynek célja a személyi számítógépeken lévő információk hatékonyabb rendszerezése, kezelése, megértése és megosztása. Számos technológiát sorakoztat fel. Készült egy saját, Java alapú lekérdező keretrendszer hozzá *Sesame*, amit később a *Virtuoso* váltott fel.

A KDE asztali környezetbe lehet integrálni. A dokumentumok keresésére és címkézésére az egyik program a Dolphin fájlkezelő. Ebben a lekérdezések grafikus felületen keresztül adhatók ki [SJ11].

[2] [Nepomuk] http://nepomuk.semanticdesktop.org/nepomuk

A Dolphin-nal a fájlokra kereshetünk többek között módosítási dátum, méret, címkék vagy értékelés alapján is. A szűréseket több lépésben is finomíthatjuk (2.2).



2.2. ábra. A Dolhpin integrált keresőfelülete szűrési feltételek megadásához a keresésekben.

Ez a felület bonyolultnak bizonyult, túlságosan is sokat kellett kattintani az egyes feltételek megadásához. A Nepomuk-ban rejlő keresési lehetőségek kényelmesebb eléréséhez egy más elrendezésű, úgynevezett *faceted browser* (2.3 ábra) fejlesztése mellett döntöttek.

Ebben egy oldalsávon adhatók meg néhány kattintással a feltételek. A panel más KDE alkalmazásokban is használható. A dokumentumokon lévő címkék szerkesztése ebben a rendszerben is egy külön megnyitható szerkesztő felületen lehetséges.

2.1.4. *Tracker*

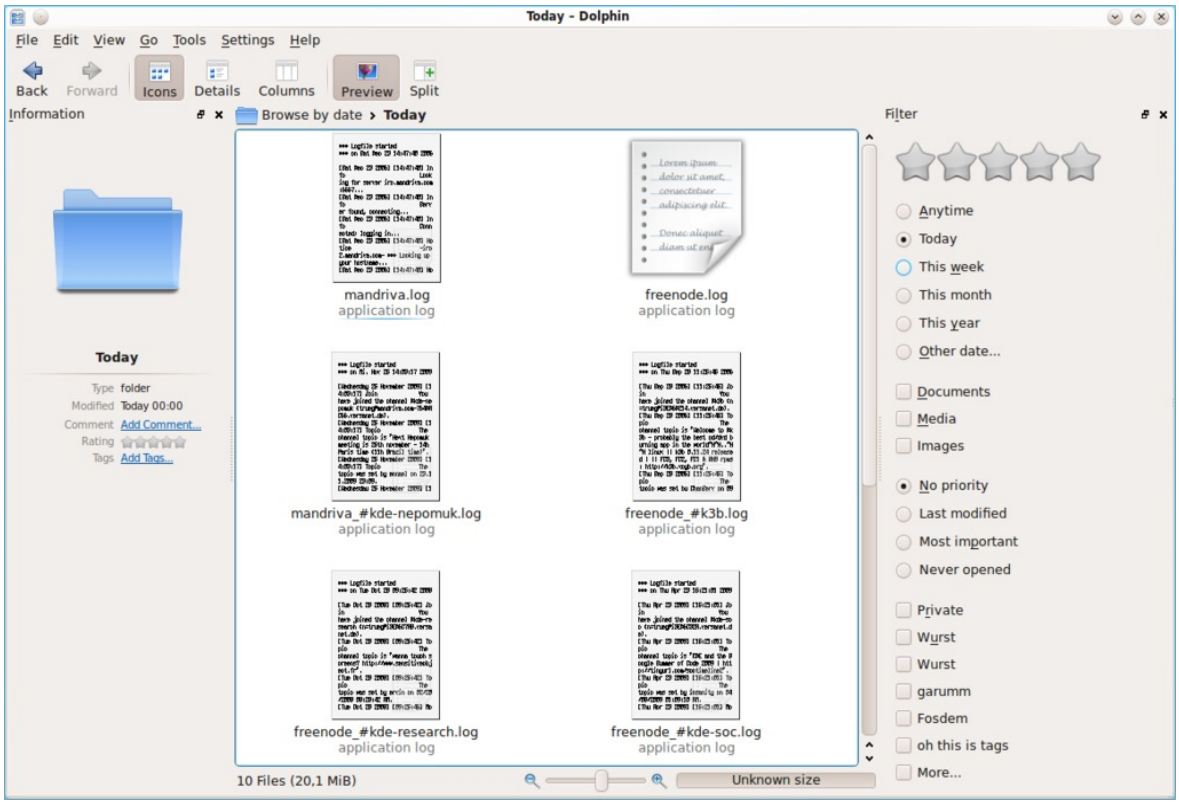
A Nepomuk elsősorban KDE asztali környezethez készült. Természetesen a Gnome-ban, mint másik elterjedt környezetben is van lehetőség. Ehhez a *Tracker*[3] használható, mely elsődleges célja szintén az asztali alkalmazások integrációja.

A Nautilus fájlböngészőbe építhető be, amely így alkalmas lesz a dokumentumok címkézésére. Kevésbé elterjedt mint a Nepomuk a KDE-s disztribúciók esetén.

2.1.5. *tag2find*

Egy kisebb program, amellyel egyszerűen adhatunk címkéket a fájlokhoz, majd azok alapján visszakereshetjük őket [4]. Windows operációs rendszerhez készült. A tálca értesítési felület részén az ikonjára kattintva megnyithatjuk a keresőmezőt. A címkék megadását automatikus kiegészítéssel segíti, felsorolja javaslatként a kapcsolódó címkéket azok gyakoriságával együtt, illetve az így megadott keresésre illeszkedő fájlokat is. Ehhez is tartozik természetesen címkeszerkesztő, illetve számos segédfunkció is (2.4 ábra).

[3] [Tracker] https://wiki.ubuntu.com/Tracker
[4] [Tag2find] http://www.tag2find.com



2.3. ábra. *faceted browser*: A keresési felület egy kényelmesebb, oldalsávos megoldása.

2.1.6. *TaggedFrog*

A TaggedFrog[5] egy általános címkézőprogram, amellyel tetszőleges fájlok vagy hivatkozások láthatók el címkékkel. Ez úgy történik, hogy a fájlokat könyvtárakba szervezhetjük, amelyeket utána egy címkével jelölhetünk meg. A dokumentumokat ellátja néhány automatikusan kiválasztott címkével is. A kezelőfelülete nagyon letisztult (2.5 ábra).

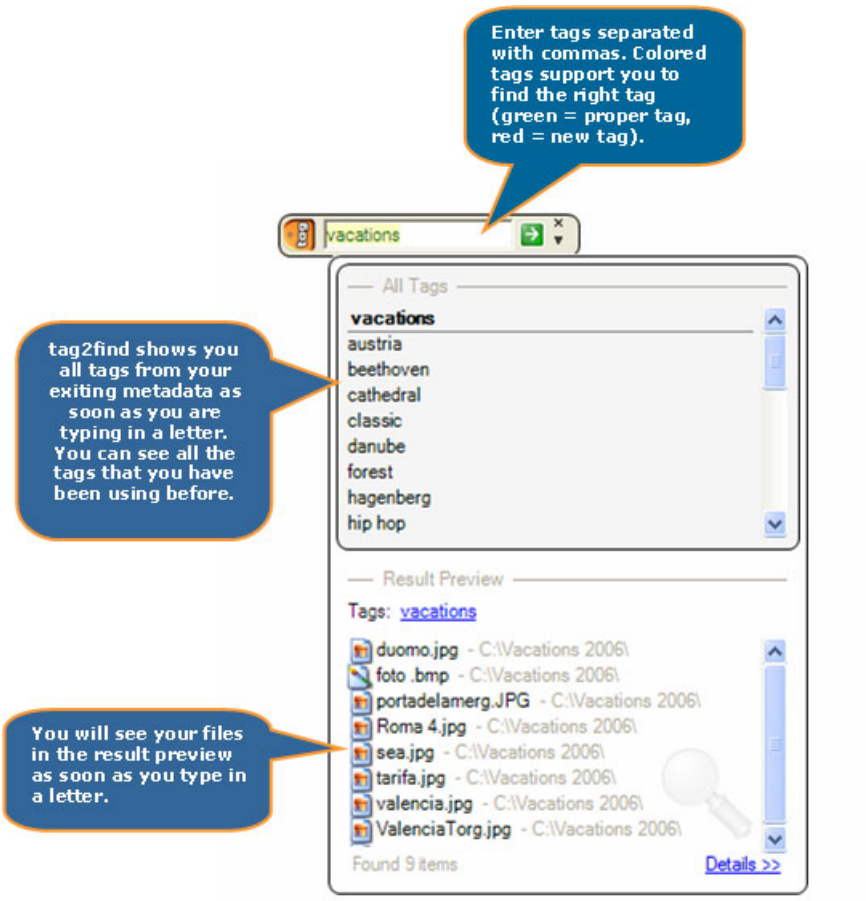
A címkéket egy címke felhőben (*tag cloud*) jeleníti meg. Ezeket sorban kijelölve szűkíthetjük a keresést, melynek eredménye az ablak alsó részén jelenik meg.

2.1.7. *OpenKM*

Az eddigi programokban a címkézésről elsősorban a felhasználó saját dokumentumainak rendszerezése kapcsán esett szó. Többfelhasználós, elosztott környezetben szintén használatos a módszer. Egyik formája az, amikor a szervezetek dokumentumainak kezelésénél a tudásmenedzsment részeként jelenik meg.

Az OpenKM (*Open Knowledge Management*) egy nyílt forráskódú elektronikus dokumentumnyilvántartó rendszert[6]. Rengeteg funkciója van; a dokumentumokat kategóriákba rendezhetjük, metaadatokkal egészíthetjük ki, illetve különböző szempontok alapján kereshetünk a nyilvántartásban. A szolgáltatásait egy webes felületen érhetjük el. Adhatunk kulcsszavakat is a dokumentumokhoz, melyeket egy kulcsszó felhőben (*keywords cloud*) fog megjeleníteni a rendszer a gyakoriságuknak megfelelően (2.6 ábra).

[5] [TaggedFrog] http://lunarfrog.com/taggedfrog
[6] [OpenKM] http://www.openkm.com



2.4. ábra. A címke alapján történő keresés menetét bemutató ábra a dokumentációból.

2.1.8. *Tabbles, folders evolved*

A Tabbles[7] egy olyan elosztott környezetre tervezett rendszer, amely a dokumentumok rendszerezését, és keresését kifejezetten címkék használatával oldja meg. A *tabble* itt egyben jelent címkét és virtuális mappát is. A címkék jelentős részét automatikusan rendeli a dokumentumokhoz. A felhasználói felülete a Windows fájlkezelő funkcióit egészíti ki, a megszokott *drag and drop* technikával is adhatunk meg címkéket.

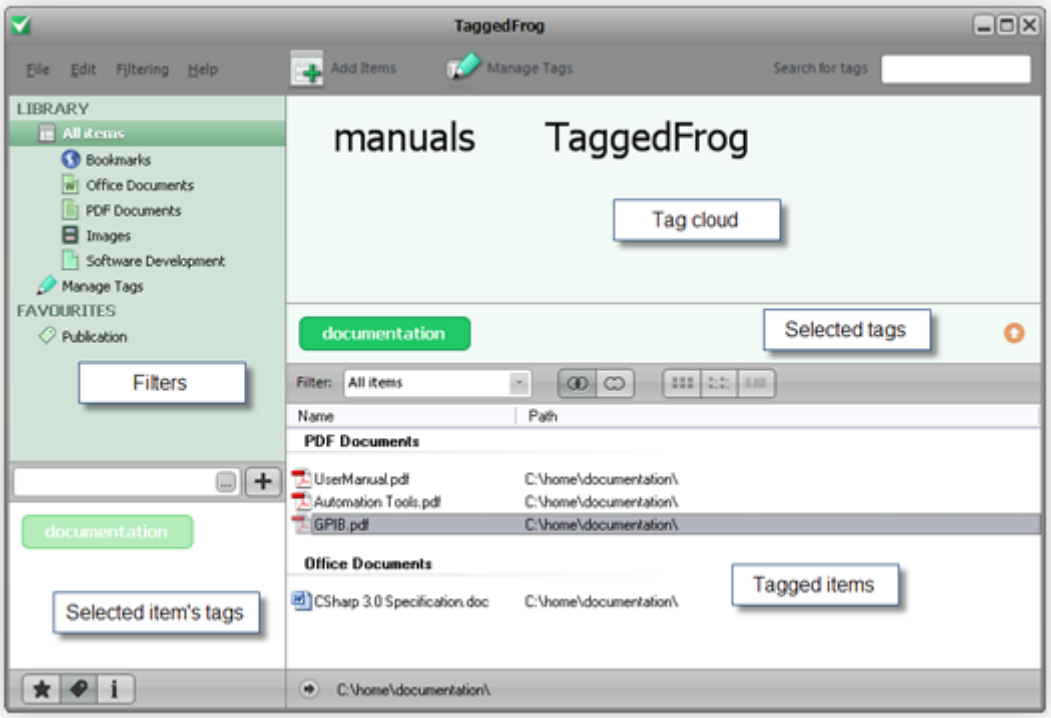
Ebben a rendszerben is a virtuális mappa koncepció a mérvadó. A rugalmasságot itt az adja a hierarchiához képest, hogy egy fájl több helyen is megjelenhet. A virtuális mappákat itt kör alakú ikonok jelölik (2.7 ábra).

A program működési elvét tekintve jelentősen nem különbözik a Windows beépített megoldásától. A létjogosultságát, esetlegesen sikerét, a kényelmesebb felhasználói felületnek köszönheti, melyet a honlap főoldalán lévő "*..what WinFS should have been.*" mondat is jelez.

2.2. *A meglévő rendszerek értékelése*

A tapasztalatok alapján a címkéző rendszerek a hagyományos rendszerekre épülnek, mint egy absztrakciós réteg. Ez az alkalmazások megvalósítása szempontjából előnyös, viszont így rejtve ezekben is megmaradnak az alaprendszer kötöttségei.

[7] [Tabbles] http://tabbles.net



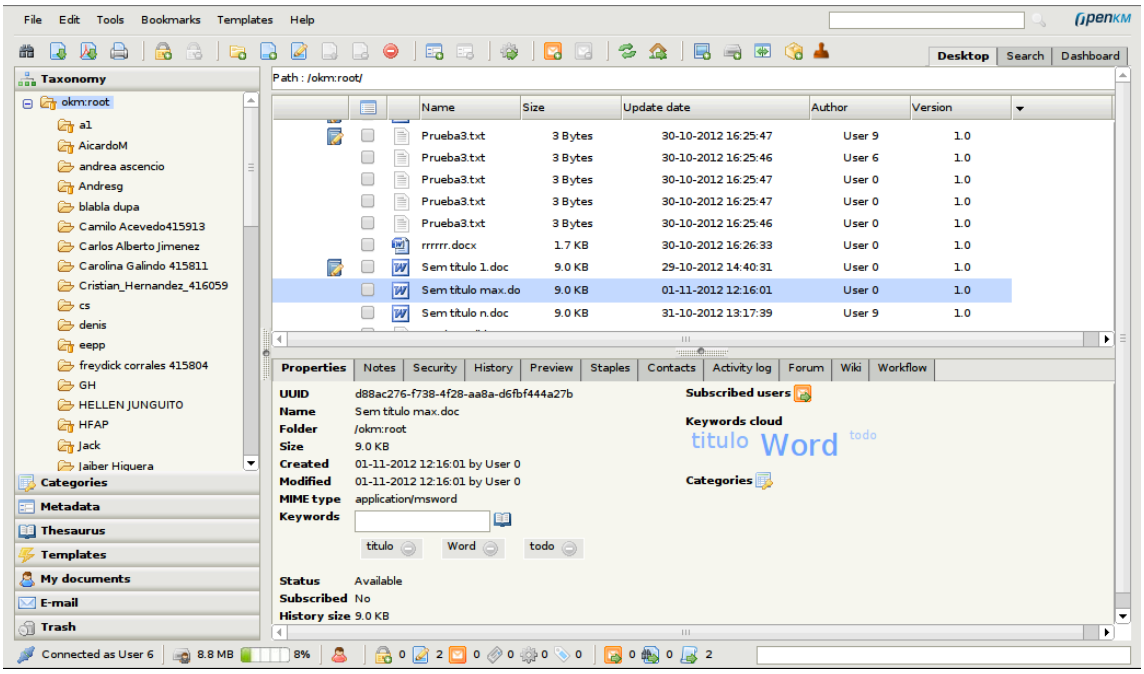
2.5. ábra. A TaggedFrog kezelőfelülete.

Számos automatizálási funkciót tartalmaznak. Ezek közé hozzátartozik a típus esetleg tartalom alapján történő címkézés, illetve a hierarchia konverziója is. Mindezek segítik, kényelmesebbé teszik a használatukat.

A címkézéshez használt felületekben is megjelennek bizonyos konvenciók. A címkék szerkesztéséhez gyakorlatilag mindenütt címkelisták vannak, azok hozzárendelését a dokumentumokhoz ezek segítségével lehet megadni. A hierarchiában megszokott módszerhez képest a felhasználó szemszögéből ez jóval nagyobb erőfeszítést igényel a rendszerezéshez.

A keresés és a címkék szerkesztése alapvetően elkülönül. Gyakran külön felületek szolgálnak ezen funkciók elérésére, ami így a felhasználó számára kevésbé lesz szemléletes.

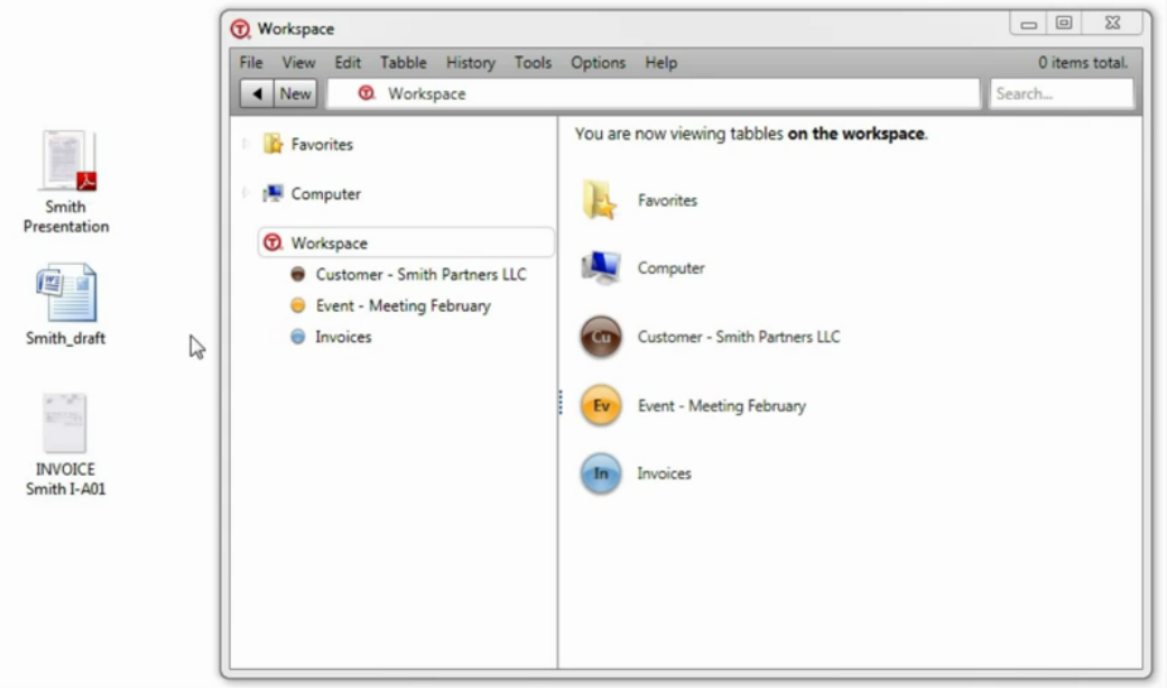
A rendszerek között az adatok átvitele körülményes, mivel nincs jelenleg széles körben elterjedt, egységes módszer a címkék, és hozzá tartozó metaadatok kezelésére. Ennek egyik fő okát a címkézési módszerekben rejlő elvi különbségek jelentik.



2.6. ábra. Az OpenKM webes kezelőfelülete.

Az új rendszer fejlesztésénél először a címkézési módszer elvi megalapozását kell megtenni. Ehhez az eddigi alkalmazásokban ismertnek vélt fogalmakat is közelebbről meg kell vizsgálni, majd ennek megfelelően megadni a szerkezeti és műveleti elemeket. Ezekről részletesebben a 3. fejezetben lesz szó.

A felhasználói felületek említett problémáira a 4. fejezetben tárgyalt újfajta felépítés és használati mód igyekszik megoldást adni.



2.7. ábra. A tabbles kör alakú ikonokkal jelzett virtuális mappái.

3. fejezet

Az új rendszer alapelvei

3.1. A modell formális leírása

Általánosságban a dokumentumokat tekinthetjük, mint tetszőleges objektumokat, a címkéket pedig mint ezek attribútumait. Az ezek közötti belső kapcsolatok elemzésének egyik fő eszköze a fogalomhálók elmélete (*Formal Concept Analysis*) [RS98] [WB04].

Az objektumok halmazát G-vel, az attribútumok halmazát pedig M-el szokás jelölni. Tekintsük továbbá az $I = G \times M$ bináris relációt (*incidence relation*). A (G, M, I) hármast kontextusnak nevezzük. A relációt táblázatos formában is szemléltethetjük. Legyen most $G = \{g_1, \ldots, g_4\}$ és $M = \{m_1, \ldots, m_6\}$ halmazok. Ezek között egy lehetséges megfeleltetést mutat a következő táblázat.

	m_1	m_2	m_3	m_4	m_5	m_6
g_1	×				×	
g_2	×			×	×	×
g_3		×	×			×
g_4	×			×	×	

Tekintsünk egy $A \subseteq G$ objektumhalmazt. Definiáljuk az

$$A' = \{m \in M | (g,m) \in I, \forall g \in A\}$$

halmazt. Hasonlóképpen definiálható a $B \subseteq M$ attribútumhalmazhoz a

$$B' = \{g \in G | (g,m) \in I, \forall m \in B\}$$

halmaz. Az $A \to A'$ és $B \to B'$ leképzéseket derivációnak nevezzük. Fogalomnak nevezzük azon (A, B) párokat, amelyekre teljesül, hogy $A' = B$ és $B' = A$. Az A halmazt ekkor attribútum-extenziónak (*extent*), a B halmazt pedig objektum-intenziónak (*intent*) hívjuk [VL10].

Fogalmak között beszélhetünk tartalmazási kapcsolatról is. Tegyük fel, hogy az (A_1, B_1) és (A_2, B_2) fogalomnak is részei a kontextusnak. Ha teljesül, hogy $A_1 \subseteq A_2$, akkor az (A_1, B_1) fogalom az (A_2, B_2) részfogalma. A tartalmazásból következik, hogy $A_2' \subseteq A_1'$. A tartalmazás egy részbenrendezést ad meg, amire teljesülnek az

$$(A_1, B_1) \leq (A_2, B_2) \iff A_1 \subseteq A_2 \iff B_2 \subseteq B_1$$

összefüggések. Jelölje $\mathfrak{B}(G, M, I)$ az egy kontextushoz tartozó összes fogalom halmazát. Az előbbi rendezési művelettel fogalom hálót (*concept lattice*) kapunk, amely egy teljes háló, mivel bármely részhalmazának van szuprémuma és infimuma [RK06].

Az összes fogalom megkeresésének több módja is lehet. A deriváció tulajdonságaiból kiindulva a lezárás műveletének (*closure*) bevezetésével kaphatjuk a következő módszert [BG06], [KL03], [KL07].

Mivel a deriváció mindkét halmazra értelmezve van, ezért azt többször is alkalmazhatjuk, így például az $A'' \subseteq G$ és $A''' \subseteq M$ halmazokat is származtathatjuk. Tekintsük az $A, A_1, A_2 \subseteq G$ halmazokat. Ezekre teljesül, hogy

$$A_1 \subseteq A_2 \Rightarrow A_2' \subseteq A_1', \quad A \subseteq A'', \quad A' = A'''.$$

Hasonlóképpen kapjuk a $B, B_1, B_2 \subseteq M$ halmazokra, hogy

$$B_1 \subseteq B_2 \Rightarrow B_2' \subseteq B_1', \quad B \subseteq B'', \quad B' = B'''.$$

Az $A \subseteq G$ részhalmazból kétszeres derivációval kapott $A'' \subseteq G$ halmazt extenzió lezártnak (*extent closure*), a dualitásnak megfelelően a $B \subseteq M$ halmazból származtatott $B'' \subseteq M$ halmazt pedig intenzió lezártnak (*intent closure*) nevezzük. Az A halmazban szereplő összes objektum közös attribútumai az A''-ban lévő összes objektumnak is a közös tulajdonságai lesznek, illetve ha egy objektum rendelkezik a B-ben lévő attribútumok mindegyikével, akkor a B''-ben attribútumokkal is rendelkezni fog.

A deriváció mintájára bevezethetjük az $X \to X''$ lezárás műveletet (*closure operator*), melyre

$$A_1 \subseteq A_2 \Rightarrow A_1'' \subseteq A_2'', \quad A \subseteq A'', \quad (A'')'' = A'',$$

illetve

$$B_1 \subseteq B_2 \Rightarrow B_1'' \subseteq B_2'', \quad B \subseteq B'', \quad (B'')'' = B''$$

összefüggéseknek is teljesülni kell. Zártnak nevezzük a halmazt, ha zárt erre az operációra nézve.

Egy (G, M, I) formális kontextus esetén ha $A \subseteq G$, akkor A'' egy extenzió lesz, ha pedig A már eleve egy extenzió lenne, akkor pedig $A'' = A$. Ez az attribútumokhalmazokra is teljesül, vagyis ha $B \subseteq M$, akkor B'' egy intenzió, és minden B intenzió esetén $B = B''$. Ez abból következik, hogy tetszőleges A és B halmazok esetén (A'', A') és (B', B'') is formális koncepciók lesznek. [GW05]

Tekintsünk egy olyan M halmazt, amin értelmezhető egy lineáris rendezés, vagyis

$$M = \{m_1 < m_2 < \ldots < m_n\}.$$

Minden $g \in G$ objektumhoz ekkor egyértelműen rendelhető egy olyan $\mathbf{m}$, n elemű bináris vektor (karakterisztikus vektor), melyben az i-edik elem értéke 1, ha $(g, m_i) \in I$, egyébként pedig 0. (A vektor itt sorvektor, mivel tulajdonképpen a reláció táblázatának sorairól van szó.)

Tetszőleges két objektumhoz tartozó $\mathbf{p}, \mathbf{q} \in \{0,1\}^n$ attribútumokat leíró vektorok között megadható egy lexikális rendezés, méghozzá

$$\mathbf{p} < \mathbf{q} \quad \Longleftrightarrow \quad \exists k, \text{ hogy } p_k = 0 \neq q_k, \text{ és } \forall j < k \text{ esetén } p_j = p_k \quad (1 \leq k, j \leq n).$$

Bevezethetjük a $<_k$ relációt is, melyben k az előbbi feltételeket kielégítő k értékek közül a legkisebb. Ezzel a relációval

$$\mathbf{p} < \mathbf{q} \quad \Longleftrightarrow \quad \exists k, \text{ hogy } \mathbf{p} <_k \mathbf{q} \quad (1 \leq k \leq n).$$

Teljesül továbbá, hogy ha $\mathbf{p} <_i \mathbf{q}$ és $\mathbf{q} <_j \mathbf{r}$, ahol $i < j$, akkor $\mathbf{q} <_i \mathbf{r}$.

Amennyiben teljesül, hogy $B_1 \subseteq B_2$, akkor abból már következik a lexikális rendezés szerinti $B_1 < B_2$ reláció is a megfelelő karakterisztikus vektorokra értelmezve.

Az összes fogalom megkereséséhez meg kell találni az összes lezártat. Az összes lehetőség leszámlálása felesleges és időigényes, mivel minden esetben ellenőrízni kell, hogy a koncepció szerepel-e már a megtaláltak között. Ennél hatékonyabb, ha a lexikális rendezésnek megfelelő sorrendben sikerül előállítani azokat.

Célszerű bevezetni egy újabb műveletet, méghozzá

$$B \oplus m_i := ((B \cap \{m_1, m_2, \ldots, m_{i-1}\}) \cup \{m_i\})''.$$

Ez tehát eltávolítja az i, vagy annál nagyobb indexű attribútumokat, hozzáveszi az i indexűt, majd kiszámítja a lezártat. A műveletre teljesülnek a következők.

- $m_i \notin B \Rightarrow B < B \oplus m_i$
- Ha B_1 egy zárt halmaz, és $B_1 <_i B_2$ akkor $B_1 \oplus m_i \subseteq B_2$, amiből következik a $B_1 \oplus m_i \leq B_2$ reláció is.
- Ha B_2 zárt halmaz, és $B_1 <_i B_2$, akkor $B_1 <_i B_1 \oplus m_i$.

A rákövetkezőségre az eddigiek a következő tételben foglalhatók össze. [BG84]

3.1. tétel. *A $B \subseteq M$ után a lexikális rendezés szerint következő B^+ zárt halmazt, ha létezik akkor a*

$$B^+ = B \oplus m_i$$

kifejezéssel kaphatjuk, ahol $m_i \in M$ az a maximális elem, amire még teljesül, hogy $B <_i B \oplus m_i$.

A fogalmak megkereséséhez tehát sorban fel kell sorolni az összes intentenziót (vagy extenziót), majd ezekből deriválással kaphatjuk a formális koncepciókat. Kiindulásképpen az üres halmaznak a lezártját kell venni, vagyis $B = \varnothing''$. A következő halmazok az alábbi algoritmusnak megfelelően állíthatók elő.

```
nextClosure(B):
    m_i = m_{i+1}
    success = false
    while success == false and m_i != m_1
        m_i = m_{i-1}
        if m_i ∉ B
            B = B ∪ {m_i}
            C = B''
            if ∄m_j < m_i, ∀m_j ∈ {C \ B}
                B = C
                success = true
        else
            B = B \ {m_i}

    return B
```

Az utolsó intenzióként magát az M-et kapjuk vissza, ami így használható leállási feltételként is.

A számítás során a részeredményeket táblázatba rendezve nyomon lehet követni az algoritmus működését. A példaként hozott kontextusra ez a következőképpen adódik.

B	m_i	$(B \cap \{m_1, \ldots, m_i\}) \cup m_i$	$C = B \oplus m_i$	$m_j < m_i$
$\varnothing$	m_6	$\{m_6\}$	$\{m_6\}$	-
$\{m_6\}$	m_5	$\{m_5\}$	$\{m_1, m_5\}$	$m_1 < m_5$
$\{m_6\}$	m_4	$\{m_4\}$	$\{m_1, m_4, m_5\}$	$m_1 < m_4$
$\{m_6\}$	m_3	$\{m_3\}$	$\{m_2, m_3, m_6\}$	$m_2 < m_3$
$\{m_6\}$	m_2	$\{m_2\}$	$\{m_2, m_3, m_6\}$	-
$\{m_2, m_3, m_6\}$	m_5	$\{m_2, m_3, m_5\}$	$\{m_1, \ldots, m_6\}$	$m_1 < m_5$
$\{m_2, m_3, m_6\}$	m_4	$\{m_2, m_3, m_4\}$	$\{m_1, \ldots, m_6\}$	$m_1 < m_4$
$\{m_2, m_3, m_6\}$	m_1	$\{m_1\}$	$\{m_1, m_5\}$	-
$\{m_1, m_5\}$	m_6	$\{m_1, m_5, m_6\}$	$\{m_1, m_4, m_5, m_6\}$	$m_4 < m_6$
$\{m_1, m_5\}$	m_4	$\{m_1, m_4\}$	$\{m_1, m_4, m_5\}$	-
$\{m_1, m_4, m_5\}$	m_6	$\{m_1, m_4, m_5, m_6\}$	$\{m_1, m_4, m_5, m_6\}$	-
$\{m_1, m_4, m_5, m_6\}$	m_3	$\{m_1, m_3\}$	$\{m_1, \ldots, m_6\}$	$m_2 < m_3$
$\{m_1, m_4, m_5, m_6\}$	m_2	$\{m_1, m_2\}$	$\{m_1, \ldots, m_6\}$	-

A második és harmadik oszlop értékeinek számítása a pszeudókódban több lépésben történik. Az utolsó oszlop mutatja, hogy az aktuális C érték intenzió-e. Amennyiben nem tudunk találni $m_j \in \{C \setminus B\}$-t, hogy fennáljon az $m_j < m_i$, akkor intenziót kaptunk. A táblázatban külön nem szerepel a legelső lépésben kapott $\varnothing$ intenzió, ezért azt még hozzá kell venni az eredményhez. Összegezve tehát a következő fogalmak adódtak.

$$\mathfrak{B}(G, M, I) = \{(\{g_1, g_2, g_3, g_4\}, \varnothing), (\{g_2, g_3\}, \{m_6\}), (\{g_3\}, \{m_2, m_3, m_6\}),$$
$$(\{g_1, g_2, g_4\}, \{m_1, m_5\}), (\{g_2, g_4\}, \{m_1, m_4, m_5\}), (\{g_2\}, \{m_1, m_4, m_5, m_6\}),$$
$$(\varnothing, \{m_1, m_2, m_3, m_4, m_5, m_6\})\}$$

A koncepcióháló a 3.1 ábrán látható Hasse-diagrammal szemléltethető.

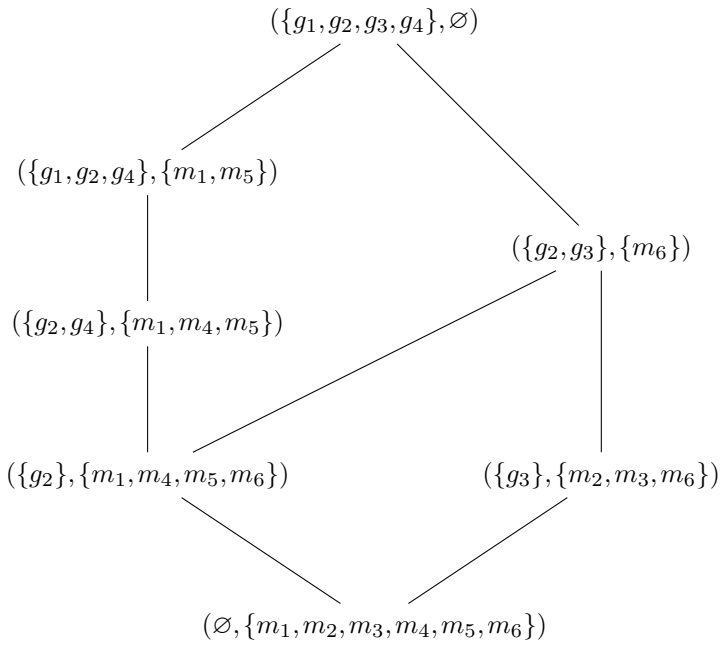
A fogalmak kereséséhez egy, a gyakorlatban jobban használható eszköze az *InClose* algoritmus [AS09].

3.2. *A kontextus felépítése*

A kontextus kialakításához szükséges elemi műveleteket is definiálni kell. Ezek teszik majd lehetővé az új elemek felvitelét, illetve azok módosítását. A konstrukcióból közvetlenül látható, hogy melyek lesznek ezek, viszont külön sorra kell venni őket, mivel ezek jelentik a struktúrához tartozó műveleti részt.

A bővítéshez a kontextus hármas felosztásának megfelelően az alábbi műveletek adhatók meg.

- *objektum hozzáadása*: A G halmaznak egy új g elemmel való bővítését jelenti, tehát $G := G \cup g$.



3.1. ábra. A fogalmak hálójának ábrázolása Hasse-diagrammal.

- *attribútum hozzáadása*: Az M halmazt bővíti egy új m-mel, vagyis $M := M \cup m$.
- *kapcsolat hozzáadása*: Egy $g \in G$ objektum látható el vele az $m \in M$ attribútummal, vagyis $I := I \cup \{(g, m)\}$.

A módosításhoz ezeknek megfelelően az alábbiak kellenek.

- *objektum eltávolítása*: Egy g objektum eltávolításához $G := G \setminus \{g\}$.
- *attribútum eltávolítása*: Egy m attribútum eltávolításához $M := M \setminus \{m\}$.
- *kapcsolat eltávolítása*: A $g \in G$ és $m \in M$ között fennálló kapcsolat eltávolításához $I := I \setminus \{(g, m)\}$.

A kontextus alakításához az összes művelet ezen elemi műveletekből építhető fel.

3.3. *Kapcsolat a hierarchiával*

Egy alkatalógusnév egy attribútumnak feleltethető meg. A fastruktúrában tárolt objektumokat egyértelműen az ezekből alkotott elérési útvonaluk azonosítja. Ez az útvonal egy rendezett név k-s. Az előző jelöléseket használva tehát itt is G halmazbeli objektumaink vannak, a "helyüket" pedig M halmazbeli jellemzők adják meg. Ez az M halmaz tehát most az aljegyzékek neveit tartalmazza. Az aktuális rendszerre nézve ezeket globálisnak kell tekinteni, egy jegyzék nevéhez tehát nem kapcsolódik ebben a modellben a hozzá vezető útvonal.

Egy $g \in G$ objektumhoz tartozó útvonal itt egy $\mathbf{p} \in M^k$ vektor lesz. A k értéke természetesen itt objektumonként változhat a fa mélységének megfelelően.

A jelentős különbségek a dokumentum elérési módjának megadása szempontjából, hogy

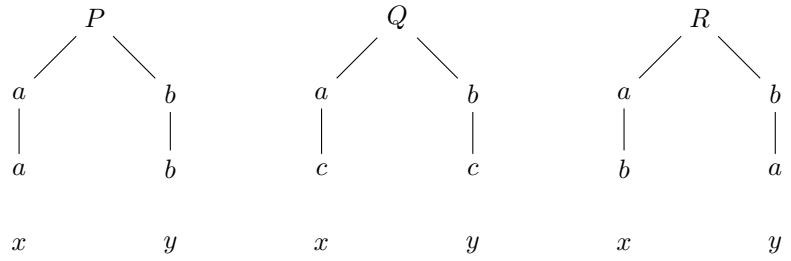
- a hierarchiában az attribútumok sorrendje kötött, míg halmaz esetén tetszőleges;

- az attribútumok halmazában nyilvánvalóan egy jellemző csak egyszer szerepelhet, míg az útvonal esetén tetszőleges sokszor.

Az átalakítás a két megadási mód között elvi szinten némi kiegészítéssel mindig egyszerűen elvégezhető. Tekintsük először azt az esetet, amikor hierarchiában vannak az objektumok, és ehhez szeretnénk ekvivalens módon egy relációt felírni. A problémát itt a nevek ismétlődése okozhatja. Az olyan triviális esetekkel nem érdemes foglalkozni, hogy például az elérési útvonal is lehet attribútum, illetve hogy a hierarchiában minden csomópont neve egyedi legyen. Az olyan átalakítások a szerencsések, ahol a lehető legkevesebb változtatásra van szükség.

Az egyértelműség a megadási módok között szigorú értelemben azt jelenti, hogy teljes egészében visszaállítható a struktúra. Kevésbé erős megkötés az, ha csak azt várjuk el az átalakítástól, hogy az útvonalon szereplő összes attribútumot megadva ugyanazt az objektumhalmazt kapjuk vissza, mint amit az útvonal kijelöl.

Legyen most $x, y \in G$ két objektum, illetve $a, b, c \in M$ attribútumok. A 3.2 ábrán látható három eset, amelynél a közvetlen átírásnál problémák lehetnek.



3.2. ábra. A relációba való átíráskor problémás esetek.

Alsó indexben jelölve a hierarchia gyökerét a következő bináris relációkat kapnánk az elérési útvonalban szereplő attribútumokat egyszerűen hozzárendelve az objektumhoz:

$$\begin{aligned} I_P &= \{(x,a),(y,b)\}, \\ I_Q &= \{(x,a),(x,c),(y,b),(y,c)\}, \\ I_R &= \{(x,a),(x,b),(y,a),(y,b)\}. \end{aligned}$$

A P hierarchiában az egyértelműséghez megkülönböztethetővé kell tenni az attribútumokat a szinteknek megfelelően. Feltételezhető, hogy a különböző szinteken lévő nevek nem azonosak, ezért ezt a többletinformációt az attribútum módosításával megőrizhetjük. Az a és b attribútumok helyett az első esetben az a_1, a_2, b_1, b_2 attribútumokkal már egyértelművé tehető a megfeleltetés.

A Q esetén a gondot az okozhatná, hogy azonos szinten vannak azonos nevek, tehát az előbbi átírás nem jelentene változást. Könnyen látható viszont, hogy bármennyi ilyen eset fordul is elő a gyökérhez közeledve biztosan lesz egy, ahol eltérés van.

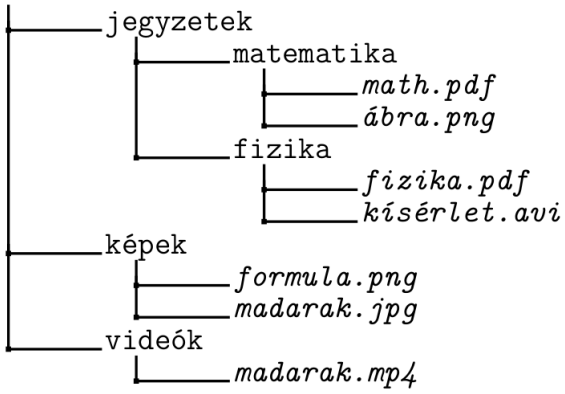
Az R hierarchiának az átírásánál az x és y objektumhoz is az $\{a, b\}$ attribútumhalmaz tartozna. A szint sorszámának attribútumhoz való hozzáadása ezt a problémát is megoldja.

A teljes visszaalakíthatóság attól függ, hogy a szintinformációt hogyan rendeltük hozzá az attribútumhoz. Amennyiben az csak megkülönböztetésül szolgál, de nem nyerhető vissza az attribútumból, akkor többféle fastruktúra is kialakítható. Például a korrigált $I_P = \{(x,a_1),(x,a_2),(y,b_1),(y,b_2)\}$ esetén is.

A relációs megadásból az átírás sokkal egyszerűbb. Ehhez csak annyit kell feltételeznünk, hogy az attribútumok rendezhetők. Általában az attribútumok nevek vagy számértékek, tehát ilyen rendezés könnyen található. Az objektumokhoz tartozó attribútumhalmazokból így egyértelműen megfeleltethetünk vektorokat, amelyek már útvonalként használhatók.

3.4. *Az átalakítás gyakorlati kérdései*

Az átalakítás az előzőek alapján mindig elvégezhető. Az viszont kérdéses, hogy az eredeti hierarchikus elrendezés valóban megfelelő volt-e. Nézzünk meg a következő egyszerű példát (3.3 ábra).



3.3. ábra. Példa a dokumentumok hierarchikus elrendezésére.

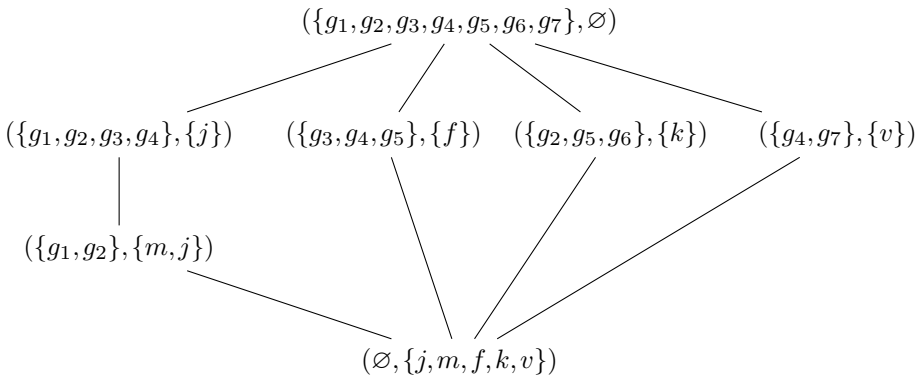
Ebben több olyan fájl is van, amely több helyre is kerülhetett volna. A jegyzéknevek itt egyediek, így közvetlenül felírható hozzá a reláció. A jegyzékek neveit, mint attribútumokat jelölje csak a kezdőbetűjük. A 3.4 táblázat az eredeti relációt bal oldalt, a hierarchiába nem illeszthető elemekkel kiegészített relációt pedig jobb oldalt tartalmazza.

Az így kapott koncepcióháló Hasse-diagramja a 3.5 ábrán látható. A hierarchia ebből úgy állítható vissza, hogy a *fizika* jegyzék a *jegyzetek* jegyzékkel egy szintre kerüljön. Ebben az elrendezésben a dokumentumok helye jobban megfeleltethető a tartalmuknak.

A halmazok közötti átfedések miatt szünnek meg a hierarchikus viszonyok. A példában kevés elem van, így az átfedések a halmazok elemszámához képest nagyok. A fastruktúrát tekinthetjük ebből a szempontból mint azt a speciális esetet, melyben nincsenek ilyen átfedések. Az elemszámok figyelembevételével a fastruktúra egy durva

	j	m	f	k	v	j	m	f	k	v
g_1: *math.pdf*	×	×				×	×			
g_2: *ábra.png*	×	×				×	×		×	
g_3: *fizika.pdf*	×		×			×		×		
g_4: *kísérlet.avi*	×		×			×		×		×
g_5: *formula.png*				×				×	×	
g_6: *madarak.jpg*				×					×	
g_7: *madarak.mp4*					×					×

3.4. ábra. Az eredeti és a kiegészített reláció.



3.5. ábra. A kibővített relációhoz tartozó koncepcióháló.

közelítését visszakaphatjuk. Amennyiben a g_5 objektum hozzávételét az f attribútumhoz tartozó objektumhalmazhoz nem tekintjük az adott halmazra nézve jelentős bővítésnek, akkor a $(\{g_3, g_4, g_5\}, \{f, j\})$ koncepciót a $(\{g_1, g_2, g_3, g_4\}, \{j\})$ részének tekintve megtartható az eredeti szerkezet.

Rendszerezés szempontjából a reláció mindenképpen jobban tükrözi a jelentést, és a hierarchikus kapcsolatokat is megőrzi, amennyiben az az adatok között fellelhető.

4. fejezet

A felhasználói felület

4.1. *A felület elemei és használatuk*

A címkézéssel a hierarchiánál általánosabb rendszerezés adható meg. Ez a kezelésben esetenként nehézségeket okozhat. Az elterjedt megközelítés, hogy a dokumentumokhoz címkelistákat adunk meg. A másik gyakran alkalmazott megoldás, hogy a címkéknek virtuális mappákat hozunk létre, és ebbe másoljuk be a dokumentumokat. Az első hátránya, hogy általában egy (esetleg több kijelölt) dokumentumnak a címkéit egyesével tudjuk csak szerkeszteni.

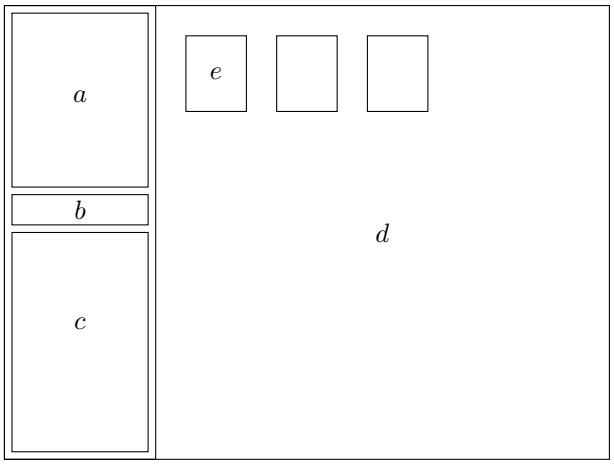
A címkézésnél jegyzékek vagy mappák helyett nézetekről (*view*) vagy hatáskörről (*scope*) beszélhetünk. Ezek használata, annak ellenére, hogy jelentés szempontjából nem hierarchiára épül a szerkezet, mégis lehetővé teszi a megszokott műveletek végrehajtását.

A felhasználói felület két fő részből épül fel. A bal oldali sávon (*tag panel*) helyezkednek el címkék és a kezelésükhöz szükséges gombok, tőle jobbra pedig a lekérdezések eredményeképpen kapott dokumentumok (*document panel*) (4.1 ábra). Ez a felosztás tehát pontosan megfeleltethető a formális leírásban szereplő M és G halmazoknak.

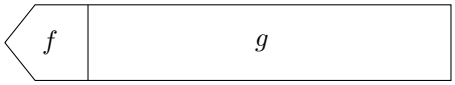
A *tag panel* három részre bontható szét. Felül van a kereséshez vagy a dokumentumokhoz tartozó címkék listája (*a.*). Ez alatt van a beviteli mező, amelybe a címkék nevét írhatja be a felhasználó (*b.*). Az alatta lévő címkelista pedig a javasolt címkék neveiből áll (*c.*). Ezekre elsősorban a kényelmesebb navigáció miatt van szükség, de a műveletek jelentős része megoldható lenne enélkül is. (Az 5.1 szakaszban ez az összetevő külön részletezésre kerül.) A dokumentumok a szokásos módon ikonok formájában jelennek meg (*e.*).

Érdemes ennek a felépítésnek megfelelően egy közbülső állapotot is beiktatni, vagyis amikor ebben az elrendezésben tudunk navigálni a hierarchikus fájlrendszerben. Ekkor az *a.* rész az útvonalakban szereplő jegyzékek nevét fogja listázni, a *b.* a gyorsabb kereséshez használható, a *c.* részre pedig az aljegyzékek listája fog kerülni. Az aljegyzéket kiválasztani a jegyzék nevének megadásával, vagy a *c.* listában lévő elemre kattintva lehet. Az aktuális (`.`) és szülő jegyzék (`..`) listázása nem szükséges, a gyökér felé közvetlenül az útvonal adott elemére kattintva lehet haladni. Jegyzékstruktúra esetén ezek a műveletek már elegendőek a navigációhoz.

Címkézés esetén a keresésben és várható (reménybeli) címkék listájában szereplő elemek két részből álló gombok (4.2 ábra). A címke bal szélén a ritkábban használandó funkciók vannak (*f.*), melyre a címke rajzolata is utal. A címke felirat része a gyakoribb feladatokhoz készült (*g.*). A vele aktuálisan elvégezhető műveletre ugyan nincs különösebb grafikus jelzés, viszont az a címke típusából és az aktuális állapotból mindig egyértelműen kiderül.



4.1. ábra. A felhasználói felület felépítése. *a*. aktuális jegyzék vagy címke lista, *b*. beviteli mező, *c*. várható címkék listája, *d*. dokumentum terület, *e*. dokumentum ikon.



4.2. ábra. A címkézéshez használt gomb felépítése. *f*. gomb a ritkábban használt funkcióknak, *g*. a felirat, és gomb a gyakoribb funkcióknak.

A működésben két fő állapot van, méghozzá amikor van kijelölve dokumentum és amikor nincs. Abban az esetben ha nincs kijelölés, az a **keresési mód**, vagyis amikor a címkelista szerkesztésével a keresést módosíthatjuk. Új címke hozzáadásához meg kell adni a címke nevét a beviteli mezőben, majd megnyomni az enter billentyűt. Gyorsabb megoldás, ha megtaláljuk a címkét a javaslatok között, mert akkor elég csak annak a nevére rákattintani. A hozzáadást követően a címke világos sárga színnel megjelenik a kereső címkék listájában (*a*.), illetve a *document panel*-en (*d*.) is változhatnak a megjelenített dokumentumok az új szűrési feltételnek megfelelően. A címke keresésből való eltávolításához csak rá kell kattintani a címkére.

Ez a lista tulajdonképpen, mint elérési útvonal funkcionál. A címkék hozzáadásával lefelé haladhatunk a hálóban, a címkék elvételével pedig felfelé. Ez a felhasználó szempontjából hasonló érzést kelthet, mint ha a hierarchiában lefelé, vagy felfelé haladna.

Negált címkéket is hozzá lehet adni a kereséshez. Ehhez a javaslati listában szereplő címkének a bal szélső részére (*f*.) kell kattintani. Ennek a szűrésnek nyilván csak olyan címkére van értelme, ami már szerepel valamilyen dokumentumon, mivel egyébként biztos, hogy nem változtatna a keresési eredményen. Eltávolítani ugyanúgy kell, mint ahogy a negáció nélküli címkét is, vagyis elég csak rákattintani.

A dokumentum kijelöléséhez rá kell kattintani a *document panel*-en a dokumentum ikonjára, és így a **szerkesztési mód** lesz aktív. A kijelölés hatására a *tag panel* is változik. A dokumentum azon címkéi, amelyek még nem szerepelnek a keresésben (ha vannak ilyenek), akkor felsorolásra kerülnek a keresőcímkék után szintén egy megkülönböztetett színnel. A beviteli mező színe is ilyenre változik, ami azt jelzi, hogy a hozzáadott címke nem a kereséshez, hanem a dokumentumhoz fog hozzáadódni.

A javasolt címkék listája itt is használható. A negált címkék dokumentumhoz való rendelésének nincs értelme, ezért ilyen esetben az magához a kereséshez adódik hozzá. A dokumentumok címkéinek nevére kattintva azokat hozzá lehet adni, vagy el lehet venni az aktuális keresésből. A címkék bal szélén lévő "×" gombbal lehet azokat eltávolítani a dokumentumról. Ha a címke a kereséshez is hozzátartozik, akkor értelemszerűen a

keresésből is kikerül egyúttal, és így az aktuális lekérdezés is változik.

A dokumentumról a kijelölést levenni úgy lehet, hogy a *document panel* olyan részére kattintunk, ahol nincs dokumentum ikon. A *tag panel* ilyenkor megtartja az aktuális keresést.

Az elvégezhető műveletekre két fontos szabály vonatkozik.

I. Kijelölt dokumentumról a rajta szereplő címkék módosításával a kijelölést elveszíteni nem lehet. Bárhogyan is adunk, vagy veszünk el címkéket, a dokumentum mindig benne lesz az aktuális szűrési feltételeknek megfelelő eredményben.

II. Egy címke nem szerepelhet többször sem a keresésben, sem a dokumentum címkéi között (mivel halmazról van szó).

Az utóbbi természetesen érvényben van akkor is, ha a keresőcímke esetleg negált lenne. Ezek a kialakításból adódóan mindig teljesülnek.

4.2. *Többszörös kijelölés*

A többszörös kijelölés úgy végezhető el, mint a klasszikus fájlböngészők esetében. Több dokumentum kijelölésekor a *tag panel*-en az egyes dokumentumok címkéiből képzett halmazok metszete jelenik meg, mint megkülönböztetett színű dokumentum címke. Ez részhalmazként ebben az esetben is tartalmazza a keresőcímkéket. A címke hozzáadása és elvétele értelemszerűen az összes dokumentumra vonatkozik. Hozzáadásnál ha a kijelölt dokumentumok valamelyikén már szerepelne az újonnan hozzáadandó címke, az nem jár fennakadással, egyszerűen ahhoz már nem adódik hozzá mégegyszer. Címkét eltávolítani csak akkor lehet többszörös kijelölés mellett, ha az minden dokumentumon szerepel, így megjelenik a címkék metszetében is.

A negált címke hozzáadására vonatkozóan a két fontos szabály alkalmazhatóságát meg kell nézni. A negált címkék hozzáadásánál előfordulhat, hogy olyan címkét szeretnénk úgy hozzáadni, hogy az nem szerepel a dokumentumok közös címkéi között, viszont szerepel valamelyik, a kijelölésben szereplő dokumentumon. Ennek hozzáadásával sérülne az a feltétel, hogy a kijelölés nem változhat. Ez nem belső ellentmondásnak az eredménye, csupán annak, hogy az áttekinthetőség és egyszerűbb kezelés érdekében nem jelennek meg külön a dokumentumok egyedi, többi kijelölt dokumentumon nem szereplő címkéi is. A második szabályt tehát csak annyival kell kiegészíteni, hogy olyan negált címke nem adható a kereséshez, amely szerepel a kijelölt dokumentumok címkéinek bármelyikén.

4.3. *Fájlműveletek*

A kialakításnak köszönhetően a fájlműveletek kisebb kiegészítésekkel megfeleltethetők a hierarchikus fájlrendszerek esetében megszokottakkal. Dokumentumot létrehozni mindig csak egy adott helyre lehet. Ami a hierarchiában útvonal, az itt címke halmaz. A hierarchiával ellentétben itt nem szükséges arra vonatkozó megszorítás, hogy egymás mellett nem lehet két azonos nevű fájl. Először ez különös lehet, viszont erre az engedményre itt azért van szükség, mivel ha üres a kereséshez használt címkehalmaz, akkor minden dokumentumnak szerepelnie kell az eredményben. Az "egymás melletti" fájlok neveire vonatkozó megkötés tehát ahhoz vezetne, hogy az egész rendszerben minden névnek különböznie kell, ami nem engedhető meg.

A hagyományos másolási műveletnek itt két megfelelője van. Az elsőnél csak a címkék változnak (*add to scope*), ténylegesen egy dokumentum marad. A másiknál adatmásolásról van szó (*copy*), vagyis a műveletet követően két azonos fájl lesz. A forrás és a cél természetesen meg is egyezhet, így itt lehetséges az egyszerű duplikálás is, amikor még a fájl neve sem változik. A mozgatás (*move*) még egyszerűbben kezelhető, mivel az teljesen analóg a megszokott művelettel. Az *add to scope* műveletnél a dokumentum címkéihez hozzáadódnak a célként megadott keresésben lévő címkék, ha azok még nem lennének rajta a dokumentumon. A negált címkékre itt is külön figyelni kell, mivel a célként megadott kereső címkék között nem lehet a dokumentumon szereplő címke negáltja. Ha ez mégis így lenne, akkor az egyik lehetőség, hogy a másolást megszakítjuk, vagy pedig levesszük a dokumentumról a fennakadást okozó címkét. *copy*-nál egy olyan új dokumentum jön létre, amely címkéi a célként adott keresés címkéi, *move* esetén pedig ezekre történik csere.

A törlési műveletnek (*remove*) is több megfelelője van. Az egyszerűbb, amikor az állományt helytől függetlenül el szeretnénk távolítani. Ennek hatására az összes előfordulási helyéről eltűnik, kikerül a fájlrendszeri nyilvántartásból, esetleg fizikálisan is törlődik. Lehetséges lenne egy olyan törlési művelet is, mely a dokumentum címkéi közül eltávolítja azokat amelyek az aktuális keresésben szerepelnek. Ennek a megvalósítása technikai problémát nem okoz, viszont felhasználás szempontjából nem tűnik érdemesnek a bevezetése. E helyett használhatóak a már bemutatott címke módosítási lehetőségek.

A fájlműveletek többszörös kijelölés esetén is ugyanúgy működnek, mint ha egyesével jelölnénk ki őket, majd ugyanazokat a műveleteket sorban hajtanánk végre rajtuk. A probléma egyedül az *add to scope* művelettel lehet, több fájl másolásánál előfordulhat, hogy csak bizonyos fájlokon szerepel olyan címke, ami a célként megadott keresésben negálva van. Ezek kezelési módjáról ilyenkor ugyanúgy dönteni kell, mint egy fájl esetében.

4.4. *További műveletek*

Az azonos fájlnevek megjelenése egy lekérdezésen belül a konstrukció következménye, viszont aránylag ritkán lehet szükség az ebből eredő lehetőségek kihasználására. A fájlnév megjelenése a címkék között nem szükségszerű (még ha gyakran praktikus is), ezért arra szűrés az eddigi műveletekkel nem lehetséges. A létrehozási, másolási és mozgatási műveletek bővíthetők egy ellenőrzéssel, amely figyeli, hogy a céldokumentumok halmazában szerepel-e már az adott névvel fájl. Ha szerepel, akkor a szokásos módon visszajelzést várhat a program a felhasználótól, hogy meghagyja az eredetit, felülírja az újjal vagy mindkettő maradjon meg. Negált címkék mellett ez az ellenőrzés csak az eredményben szereplő dokumentumokra érvényes.

Az olyan alapvető művelet, mint például a dokumentumok átnevezése, megnyitása itt külön nem került részletezésre. Ezek hasonlóan képzelhetők el, mint a már meglévő rendszerekben. A címkék nevének változtatására is szükség lehet. Ehhez a művelethez sem kell a felületen változtatni, egy felugró menüvel elérhetővé tehető ez a funkció is.

5. fejezet

A keresést támogató kiegészítő funkciók

Az alapvető keresési mechanizmust érdemes segédfunkciókkal kiegészíteni. Ezeknek a célja, hogy a felhasználó egyszerűbben, rövidebb idő alatt tudja majd megtalálni a keresett dokumentumokat. Alapvetően olyan funkciókról van szó, melyek a hagyományos rendszerek némelyikében is megtalálhatók, viszont címkézett nyilvántartás böngészésénél is nagyon hasznosak lehetnek.

5.1. *A rendszer által javasolt címkék listája*

A dokumentumok közötti navigálás a címkézés esetén jóval több irányba lehetséges, mint a jegyzékstruktúránál. A címke kiválasztása a nevének direkt megadásával mindig lehetséges, viszont nem várható el a felhasználótól, hogy minden címkét ismerjen, és a kereséshez tartozóakat mindig be is gépelje. A könnyebb kezelhetőség érdekében célszerű javaslatlistákat összeállítani. Ezt lehetőleg úgy kell megtenni, hogy a felhasználó számára aktuálisan szükséges következő címke valahol a lista eleje felé szerepeljen.

5.1.1. *Az összeállítás szempontjai*

A javaslatlisták összeállításának módja döntő fontosságú. Ennek az összeállításához többek között az alábbi összetevőket együttesen kell figyelembe venni.

- *gyakoriság*: Az adott címke előfordulásának gyakorisága az egész adatbázisra nézve. Ez gyorsan lekérdezhető, mivel az adatbázis közvetlenül is tárolja. A feltételezés az, hogy a gyakrabban használt címkék jelzik a nagyobb dokumentumcsoportokat, és a felhasználó, mint ahogyan a hierarchiában is fenntről lefelé szeretne haladni.
- *hatáskör figyelembevétele*: A keresőcímkék együttesen meghatároznak egyfajta lokalitást. A címkék jelentősége ennek a függvényében is változik. Olyan címkéket választani, amelyek nem szerepelnek az aktuális dokumentumok címkéi között keresésnél biztosan nem érdemes. Az ezekre való szűrés a dokumentumok számától függően hosszadalmasabb lehet.
- *legutóbbi dokumentumok címkéi*: A dokumentumok egy részére a felhasználók időszakosan keresnek. A nemrégiben használt dokumentumokra biztosan nagyobb valószínűséggel lesz szükség, mint egy pár héttel korábbira, így ezek címkéit is elöl érdemes szerepeltetni.

- *legutóbbi címke*: Felhasználói szempontból a címkék megadási sorrendje is fontos. A gyakori dokumentumok címkéit nem feltétlenül olyan sorrendben kapjuk, ahogyan az a felhasználónak kényelmes, ezért külön naplózni kell a címkék megadási sorrendjét is a keresésnél.
- *fogalmi háló*: A fogalmi háló is támpontot adhat. A keresésnél tulajdonképpen a Hasse-diagram élein haladunk lefelé (címke hozzáadásánál) vagy felfelé (címke elvételénél). A hatáskör figyelembevételével tulajdonképpen a lefelé vezető éleket kell megkapnunk, viszont ha előzőleg kiszámítottuk a háló bizonyos részeit, akkor gyorsabb, de pontatlanabb becslést kaphatunk rá.
- *kivételes megkötések*: Ide tartozhat például az, hogy ha egy dokumentumon még nincs címke, akkor a javaslatok között első helyen a fájl neve és kiterjesztése kell hogy szerepeljen. Bevezethetünk tiltásokat is, például olyan címkékre amelyeket nem szeretnénk, hogy a felhasználó számára megjelenjenek. Maga a felhasználó is adhat ilyen megkötéseket, például ha tudja, hogy milyen címkékre lesz szüksége a közeljövőben, azokat könyvjelzőként rögzítheti.
- *aktuális mód*: Attól függően, hogy keresésről, vagy a dokumentum címkéinek szerkesztéséről van-e szó különbözhetnek a javaslatok.

5.1.2. *Az összeállítás módja*

Az előbbi szempontokat érdemes tisztán, egymástól függetlenül kezelni. Mindegyik szempont meghatároz egy általa javasolt sorrendet. Ezt teheti például egyszerűen felsorolva a címkéket, vagy általánosabban egy pozitív egész számot rendelve hozzájuk a szerint, hogy hányadik helyre szeretné, hogy kerüljön. (Itt azonos sorszámhoz tehát tartozhat több címke is, vagy lehetnek olyan sorszámok, ahol nincs egy sem.) A címkéken szereplő sorszámokat az után a szempontoknak megfelelően súlyozással összegezhetjük. Az így kapott értékek szerint növekvő sorrendbe rendezve a címkéket előáll a javaslatlista.

A súlyozáson túl a megkötésekkel is számolni kell. Az előre megadott címkéket a helyükön kell hagyni. Továbbá hiába szerepelne az utóbbi időben használt címke elől, ha az a hatáskör miatt nem lehetne a javaslatok között. A súlyok módosítása helyett ezek is inkább szűrési feltételekként használhatók.

5.1.3. *Hatékonysági mérőszámok*

A javaslatok használhatóságának mértékére nincsenek általános mérőszámok, mivel a felhasználói szokások jelentősen befolyásolják. Bevezethetjük viszont a következő értékeket, amelyek segítenek felmérni, hogy mennyire felel meg a működés az elvárásoknak.

- *dokumentum visszakeresési idő*: A dokumentumnyilvántartás egyik fő célja minimalizálni ezt az időt. Ez az érték mutatja, hogy egy dokumentumot a keresés elkezdésétől számítva mikor sikerült megtalálnia a felhasználónak.
- *visszalépések száma*: Ideális esetben a felhasználó célirányosan, a keresés fokozatos szűkítésével halad előre. Amennyiben ez így nem sikerül meg kell számolni, hogy a keresés során mennyiszer kellett címkéket eltávolítani a keresésből.
- *szükséges kattintások száma*: Fontos kényelmi szempont, hogy ha több lehetőség is van egy dokumentum felkutatására, akkor a felhasználó ezek közül azt az utat válassza, amelyik legkevesebb energiabefektetéssel jár a részéről. Ennek egyik mérőszáma, hogy mennyiszer kellett kattintania.

- *szükséges begépelt karakterszám*: Az előzőhöz hasonló mérőszám. Az átlagfelhasználó számára a gépelés kényelmetlenséget okozhat, ezért jobb ha ez is minimális.
- *a felhasznált javaslatcímke sorszáma*: Az, hogy egy címke szerepel a javaslatlistában még csak fél siker. Lehetőleg ott kell lennie, ahol a felhasználó kényelmesen eléri azt. (Ez lehet akár a lista eleje, de vizsgálandó, hogy a lista melyik részét is nézik meg először a felhasználók.)

5.1.4. *Alkalmazkodás a használati módhoz*

A nyilvántartás adatainak és a felhasználó szokásainak megfelelően folyamatosan optimalizálnia kell a programnak a javaslatlista összeállítási módját. Ez elsősorban a súlyok módosítását jelenti. A különböző szempontok szerinti javaslatlistákat össze kell vetni a választott címkével, majd annak a szempontnak a súlyát növelni, amelyiknél elöl szerepelt.

5.2. *Dokumentumok rendezési módja*

Az előbbiekben bemutatott javaslatlistákat, az ott felsorolt szempontok alapján a dokumentumokhoz is össze lehet állítani. A dokumentumokhoz esetében viszont egyéb, hagyományosabb módokra is szükség van. A dokumentumoknak a következő jellemzői szolgálhatnak például rendezési szempontként.

- *név*: Ez a dokumentumok sorbarendezésének talán a legtermészetesebb módja. Általában lexikális rendezésről van szó, bár a kis- és nagybetűk, illetve a dokumentumok nevében szereplő számértékek miatt több változat is elképzelhető.
- *típus*: A keresésben a típus címkeként való megadása is gyakran segíthet, viszont ha egy olyan hatáskörünk van, melyben együtt szeretnénk átlátni több különböző típusú dokumentumot, akkor típus alapján szükséges rendezni azokat.
- *módosítás dátuma*: A címkéknél említett legutóbbi használat szerinti rendezésre a dokumentumoknál közvetlenül is szükség lehet.
- *méret*: Címkékre ez ilyen formában egyáltalán nem értelmezhető, viszont dokumentumok kezelésénél nagyon hasznos tud lenni.

5.3. *Szűrés név alapján*

Gyakran alkalmazott művelet a reguláris kifejezésekkel való szűkítés. A címkék és dokumentumok nevére is alkalmazható. Bonyolultabb kifejezések megadására inkább csak a dokumentumok esetében lehet szükség.

Ez a szűrési mód a címkékkel való szűrés mellett használható. A kettő között nincs sorrendi kötöttség, így a különböző típusú szűrési feltételek egymástól függetlenül, tetszőlegesen alakíthatók.

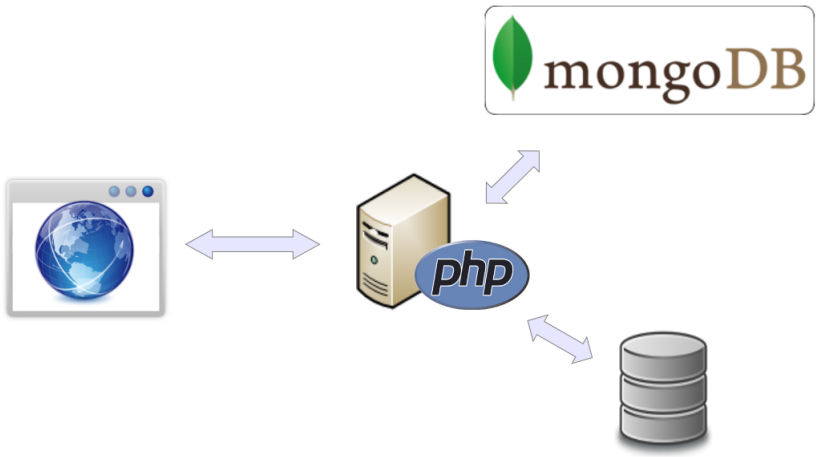
6. fejezet

Megvalósítás webalkalmazásként

A nyilvántartás prototípusának webalkalmazásként való elkészítésének érvei;

- A webalkalmazások moduláris felépítése jól illeszthető volt a feladathoz.
- A felhasználói felület kialakításához az elemek már rendelkezésre álltak. Tehát kevesebb tennivaló volt a képek megjelenítésével, a panelek elrendezésével.
- A dokumentumokkal és címkézéssel kapcsolatos adatok értelemszerűen adatbázisban tárolhatók. Az eldöntendő kérdés csupán csak az volt, hogy milyen adatbázisban és hogyan kerüljön megvalósításra, nem kellett különösebben foglalkozni az adatok tárolásának és lekérdezésének technikai kérdéseivel.
- A megoldás ilyen formában kevésbé platformfüggő mind felhasználói mind pedig a telepítés szempontjából. Felhasználói oldalról ez azt jelenti, hogy működik az elterjedt webböngészőkben. A felhasznált szoftverek szabadon elérhetők az operációs rendszerek többségén, így a szerver oldal is könnyen kialakítható.
- A felhasználói tesztek egyszerűbben megoldhatók, mivel nem szükséges külön programot telepíteni a használatához.
- A rendszer célkitűzéseit tekintve használható lenne online lekérdező felületként is. Az első változat így mintául szolgálhat egy későbbi nagyobb rendszer számára is.

A prototípus felépítése követi a webes környezetben megszokott kliens-szerver architektúrát (6.1 ábra).



6.1. ábra. A webalkalmazás felépítése. Bal oldalt látható a webböngésző alkalmazás, középen a webszerver a PHP értelmezővel ami csatlakozik az adatbázishoz és az állományok tárolójához.

Áll egyrészt egy **dokumentum tároló**ból a szerver oldalon. Ez lehet akár egy jegyzék is. Funkcionálisan mindenképpen el kell különülnie a címkék és egyéb metaadatok tárolásától. A feladata az, hogy a dokumentum azonosítójához visszaadja a hozzátartozó állomány tartalmát. (Fájlnév ebben már nem szerepel, mivel az átkerült a metaadatok közé.)

A dokumentumok és címkék adatainak tárolásához a **MongoDB** [KC10] használata kézenfekvő volt, mivel az abban kialakított dokumentumfogalom természetes módon, változtatások nélkül illeszkedett a metaadatok tárolási és lekérdezési formájához. Nyílt forráskódú program, és nagyon jól dokumentált. Rugalmas adatkezelést tesz lehetővé, illetve számos egyéb előnyének köszönhetően jó választásnak bizonyult.

A webalkalmazások központi eleme a **webszerver**. Ennek kell felügyelnie az adatbázis és a tároló működését, illetve ki kell szolgálnia a kliensek kéréseit. Mivel (mint az majd a későbbiekben kiderül) ez csak egy nagyon vékony réteget jelent, ezért az Apache szerveren PHP szkriptekkel készült el. A későbbiekben célszerű lehet áttérni a *NodeJS* használatára, mivel így teljesen homogén webalkalmazás kialakítására van lehetőség. Ez itt azt jelentené ekkor, hogy mind a kliens mind pedig a szerver oldalon futó alkalmazások JavaScript-ben készülhetnének el, és a JSON dokumentumokat sem kellene konvertálni.

A **webböngésző** látja el az összes kliens oldali feladatot. Ez alakítja majd ki a felhasználói felületet a lekérdezésekhez, továbbítja a kérést a szerver felé, majd ez jeleníti meg az eredményeket.

6.1. *A MongoDB alkalmazása*

A MongoDB az adatokat JSON[1] dokumentumok formájában kollekciókban tárolja. Jelen alkalmazáshoz a következő kollekciókra volt szükség.

- `expressions`: A címkék nevei és gyakoriságuk szerepel ebben a kollekcióban. Mindegyik kifejezésnek van egy egyedi azonosítója, így a dokumentumból elég ezt megadni mint hivatkozást.

  Példa néhány kifejezésre:

```
{
    "_id" : ObjectId("50741bd3e0656ab901000002"),
    "value" : "jpg",
    "frequency" : 8
}

{
    "_id" : ObjectId("50741bdbe0656a1103000000"),
    "value" : "photo",
    "frequency" : 5
}

{
    "_id" : ObjectId("50741c13e0656abb01000000"),
    "value" : "Miskolc",
    "frequency" : 4
}
```

[1] [JSON] http://www.json.org

- `documents`: A dokumentum itt valóban a dokumentumot is jelenti. Ez fogja össze az állományhoz tartozó összes metaadatot.

  Példa egy dokumentumra, amelyen az előbbi három kifejezés szerepel mint címke:

```
{
    "_id" : ObjectId("50741bb6e0656aba01000005"),
    "name" : "Miskolc.jpg",
    "type" : "image",
    "tags" : [
        "50741bd3e0656ab901000002",
        "50741bdbe0656a1103000000",
        "50741c13e0656abb01000000"
    ],
}
```

Egy újonnan felvitt dokumentumon először még nem szerepel semmilyen címke, ezért elegendő csak a nevének és a típusának a megadása. Amikor konkrétan egy új állomány felvitelére kerül sor, akkor ezeket az adatokat először a MongoDB kapja meg. A beszúrás során kap egy egyedi azonosítót, amit visszaad a dokumentum tárolónak, hogy olyan néven tárolja le az állományt. Az előbbi példában bemutatott dokumentum felviteléhez a következő parancsot kell kiadni:

```
db.documents.insert({
    "name": "Miskolc.jpg", "type": "image", "tags": [] }
)
```

A dokumentum módosításához ismerni kell az egyedi azonosítóját (`_id`). Ennek hiányában nem tudjuk egyértelműen kijelölni a dokumentumot. A következő műveletekre lehet szükség a metaadatok módosításához.

- *Név vagy típus módosítása*: a két művelet a parancs kiadási módja szempontjából azonos. Az előbbi dokumentumot például a `Miskolci Egyetem`-re átnevezni a

```
db.documents.update(
    { "_id" : ObjectId("50741bb6e0656aba01000005") },
    { "$set" : { "name" : "Miskolci Egyetem" } }
)
```

  paranccsal lehet.

- *Címke hozzáadása*: A címke hozzáadásakor először meg kell keresni a kifejezés azonosítóját, amivel majd címkézni szeretnénk a dokumentumot. Amennyiben ez nem sikerült, mert még a kifejezés nem szerepel az `expressions` kollekcióban, akkor az először létre kell hozni, majd a létrehozás során kapott azonosítóval kell visszatérni.

  A `tags` itt egy halmaz. Az elemek egyediségéhez nem feltétlenül szükséges belső integritásellenőrzés, mivel ha a felviteli módban megadjuk, hogy olyan címkét ne vigyen fel, amelyik már szerepel, akkor ez így mindig teljesülni fog. A parancs, amivel egy új címkét fel lehet vinni:

```
db.documents.update(
    { "_id" : ObjectId("50741bb6e0656aba01000005") },
    { "$addToSet" : { "tags" : "50741bd3e0656ab901000002"}}
)
```

- *Címke törlése*: A címkék törlése szintén egyszerűen megoldható. Itt is először a kifejezés azonosítóját kell megkeresni. Ennek a rendszer rendeltetésszerű használata mellett minden esetben szerepelnie kell az adatbázisban. Az előbbiekben felvitt címkét törölni a

```
db.documents.update(
    { "_id" : ObjectId("50741bb6e0656aba01000005") },
    { "$pull" : { "tags" : "50741bd3e0656ab901000002" } }
)
```

  parancs kiadásával lehet. A kifejezéseknél azért is célszerű nyilvántartani a gyakoriságukat, mert így azt csökkentve a törlést követően látszik, hogy arra még szükség van-e egyáltalán az adatbázisban. Az olyan kifejezéseket, amelyek egy dokumentumon sem szerepelnek címkeként elvileg rögtön lehetne is törölni, viszont felhasználó szempontjából jó, ha ideiglenesen ezek még bennemaradnak a rendszerben. A jegyzékstruktúrában ez tulajdonképpen az üres jegyzéknek feleltethető meg.

A lekérdezésnek a dokumentumok és címkék viszonylatában két fő iránya lehet:

- *Adott címkékhez tartozó dokumentumok*: Ez itt most a tipikus értelemben vett keresés, vagyis amikor az adott címkékkel ellátott dokumentumok halmazát szeretnénk visszakapni. Ehhez először a címkék azonosítóit kell visszakeresni. Az ezekből összeállított tömb, már követlenül fel is használható a lekérdezésben. A `["jpg", "fénykép"]` címkékhez tartozó dokumentumok kereséséhez a

```
db.documents.find( { "tags" : { "$all" :
[ "50741bd3e0656ab901000002", "50741bdbe0656a1103000000" ]
} } )
```

  parancs szükséges.

- *Adott dokumentumhoz tartozó címkék*: A dokumentum azonosítója alapján lehet a címkék azonosítóit lekérdezni, például:

```
db.documents.find(
    { "_id" : ObjectId("50741bb6e0656aba01000005") } ).tags
```

  Ebből az azonosítóneveket például a

```
db.expressions.find(
  { "_id" : ObjectId("50741bd3e0656ab901000002") } ).value
```

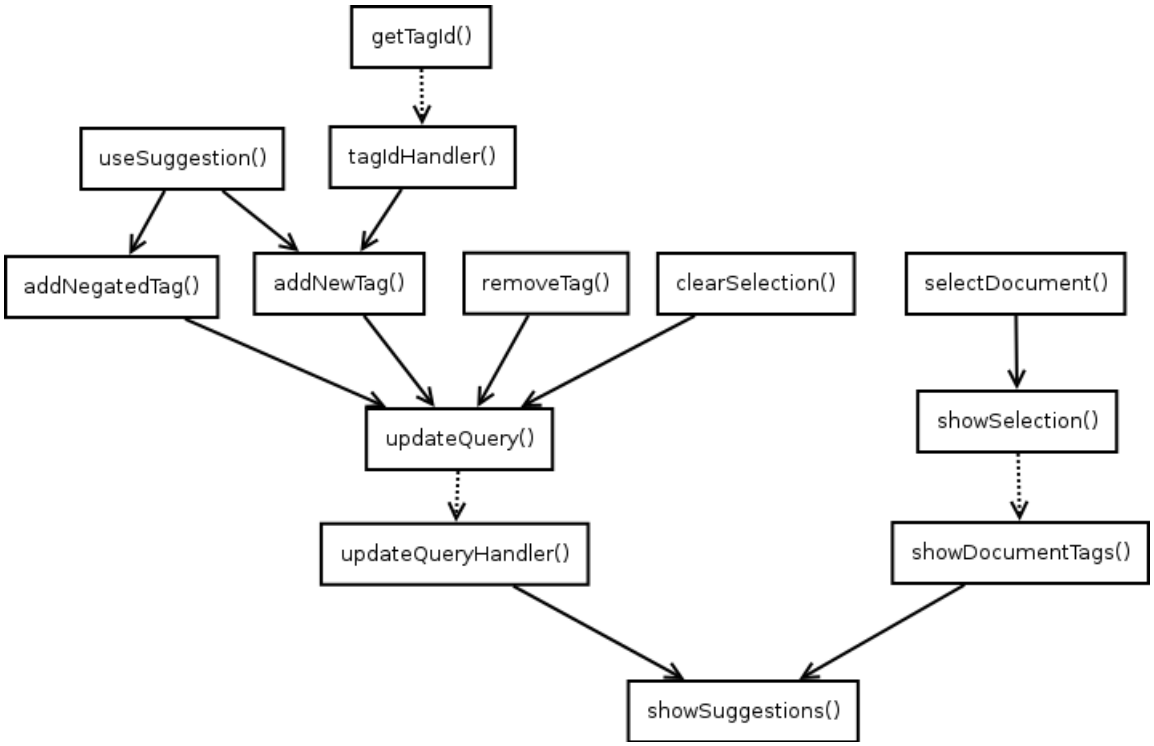
  lekérdezéssel kaphatjuk.

6.2. *A webalkalmazás működése*

Egy művelet végrehajtása általában több réteget is érint. A kliens és a szerver között a kommunikáció aszinkron kérések formájában valósul meg. A JavaScript kliens kérést küld a szerver felé, illetve megadja, hogy a választ majd melyik függvény kezelje le. Ezt a `request.js`-ben lévő `sendRequest` függvény végzi. A PHP-t és a MongoDB-t egy modul kapcsolja össze [SF12].

A 6.2 ábrán a fontosabb JavaScript-ben elkészült függvények, illetve azok kapcsolataik szerepelnek. A nyilak a használati viszonyokat jelzik, tehát fenntről lefelé haladva követni lehet a függvényhívásokat is. A pontozott vonal arra utal, hogy ott az adott függvény először egy kérést intézett a szerverhez, majd a választ a soronkövetkező függvény kezelte le.

A kéréseket a szerveroldalon a `query.php` szkript dolgozza fel, és ez válaszol rájuk. A kérések formájáról az egyes műveletek végrehajtása kapcsán lesz majd szó.



6.2. ábra. A lényegesebb JavaScript függvények és kapcsolataik.

Új címkét megadni a beviteli mezőn keresztül vagy a javaslatgombokra kattintva lehet. A beviteli mező a `getTagId` függvényt hívja meg. Ez az adott címke nevét küldi el a szerverhez, és válaszként annak az azonosítóját várja, vagy azt, hogy nincs még adott címke. A kérésben ekkor egy kötelező `expression` paraméter kell, hogy legyen. E mellett még lehet egy `mode` paraméter is, amelynek értékül az `autocreate`-et megadva automatikusan létrehozza a kifejezést, ha az még nem létezne. A válaszüzenetet a `tagIdHandler` dolgozza fel. Ez minimális ellenőrzést követően meghívja az `addNewTag` függvényt. A kijelölésnek megfelelően ez beállítja a bal oldali panelt, majd az így összeállított új címkehalmaznak megfelelően az `updateQuery` függvénnyel ismét kérést indít a szerver felé. Az aktuális címkehalmazt a dokumentum objektumban (DOM[2]) tárolja, amikor a címkék adataira van szüksége, akkor ebből keresi ki a neveket és azonosítókat.

[2][DOM] http://www.w3.org/DOM

A kérésben a `tags` paraméterrel a címkék azonosítóinak vesszővel elválasztott listájaként adja meg a lekérdezéshez szükséges címkehalmazt. A válaszban a dokumentumok adatait illetve a javaslatokat kapja JSON dokumentumként. A dokumentumok ikonjait megjeleníti, majd a javaslatlista összeállításához meghívja a `showSuggestions`-t.

A címkék listából való kiválasztását a `useSuggestion` függvény kezeli. A negált címkéket az `addNegatedTag` függvény helyezi el a panelen, majd az `addNewTag`-hez hasonlóan az `updateQuery` függvény meghívásával jelzi, hogy új lekérdezés szükséges. A lekérdezés negált címkék esetén csak annyiban módosul, hogy a címkék azonosítói elé egy minusz jel kerül.

Egy dokumentumra kattintva a `selectDocument` függvény kezdeményezi a kijelöléssel járó teendők végrehajtását. Erre külön a linkek kezelése miatt van szükség, a `showSelection` függvény fogja majd a dokumentum adatait lekérdezni, illetve a megjelenítést átalakítani. Az adatok lekérdezése a szervertől `document` paraméter megadásával lehetséges, melyben a kijelölt dokumentum azonosítóját kell átadni. A válasz, a lekérdezéshez hasonlóan, itt is egy JSON dokumentum, amelyik szintén két részből áll: a címkék adataiból, illetve a javaslatokból. A `showDocumentTags` ez alapján alakítja át a megjelenített címkéket, majd a `showSuggestions` függvénynek adja át a várható további címkék listáját. Kijelölt dokumentumhoz a címke hozzáadását szintén az `addNewTag` függvény végzi. A kérésben ekkor az `add` paraméterben adható meg a címke azonosítója, a `to` paraméterben, pedig, hogy melyik dokumentumra vonatkozik. A `removeTag` függvény törli a címkéket a dokumentumról. Ennek a két paramétere a `remove` és a `from`, amelyben itt is azonosítóikkal lehet megadni, hogy melyik címkét kell eltávolítani melyik dokumentumról.

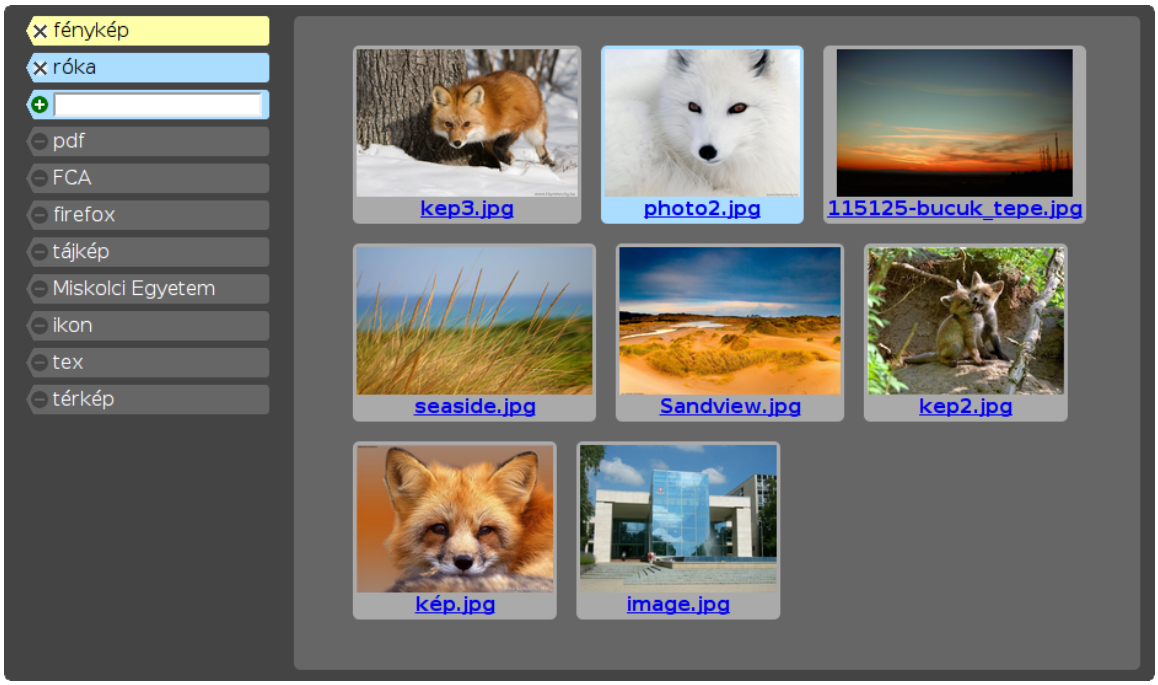
6.3. *További fejlesztési lehetőségek*

A megvalósított webalkalmazásról látható egy kép a 6.3 ábrán. Az alapvető keresési és címkemódosítási műveletek már elérhetők. A fejlesztés célja ez esetben egy ilyen prototípus létrehozása volt. Ahhoz, hogy ebből egy valódi dokumentum-nyilvántartó alkalmazás lehessen a következőkre lenne még szükség.

- A dokumentumok felvitele webes felületen keresztül még nem megoldott. Ennek előfeltétele, hogy legyen felhasználókezelés, tehát hogy be lehessen jelentkezni és csak a jogosultságoknak megfelelő műveletek legyenek elvégezhetők.
- A felület jelenleg csak egy dokumentum kijelölését támogatja, illetve mivel nem lehet több ilyen keresőt egymással összekapcsolva megnyitni egymás mellett, ezért a hagyományosabbnak mondható másolás és áthelyezés jellegű funkciók hiányoznak. A kialakítás egyik előnye éppen ezekben, a hasonló címkézést használó rendszerekből hiányzó átcímkézési módokban lenne.
- Rengeteg tesztelésre lenne még szükség ahhoz, hogy ebből egy többfelhasználós rendszer lehessen. Ebben jelentős szerep kell hogy jusson a felhasználói teszteknek is, mivel fontosak az újszerű felülettel kapcsolatos visszajelzések.

A webalkalmazások használatához elengedhetetlen, hogy a felület helyesen működjön az elterjedt böngészőkben. A fejlesztéshez *Chromium*, *Firefox* és *Opera* böngészők álltak rendelkezésre, ezekben került sor a tesztelésre. Az *Internet Explorer* böngészők esetében komoly problémák jelentkeztek. Ilyen volt például, hogy az eseményhez tartozó element az `scrElement`, míg más böngészők ezt `target` néven kezelik.

Szintén gond volt, hogy a teszteléshez használt Internet Explorer verzióban nincs implementálva a `getElementsByClassName` JavaScript függvény. A címkék listáját a



6.3. ábra. Kép az elkészült webalkalmazásról.

dokumentummodellből a `class` attribútumon keresztül lehet kigyűjteni, ezért ez is szükséges lenne a helyes működéshez. A JSON dokumentumok kezelése további eltéréseket mutatott. A weboldal stílusának kialakításnál is jelentkeztek hasonló inkompatibilitási problémák.

Az ilyen jellegű hibák megkeresése és kijavítása nem túl bonyolult, viszont időigényes. Egy éles rendszer fejlesztéséhez ezt a szakaszt nyilván nem szabad kispórolni. A cél a jelen webalkalmazás esetén csak az volt, hogy az új felületi elemek használatával kapcsolatban minél hamarabb, minél több tapasztalat lehessen.

Fejlesztési lehetőségként megjelenik még többek között a saját adatbázismotor készítése is, mellyel részletesebben a 7. fejezet foglalkozik.

7. fejezet

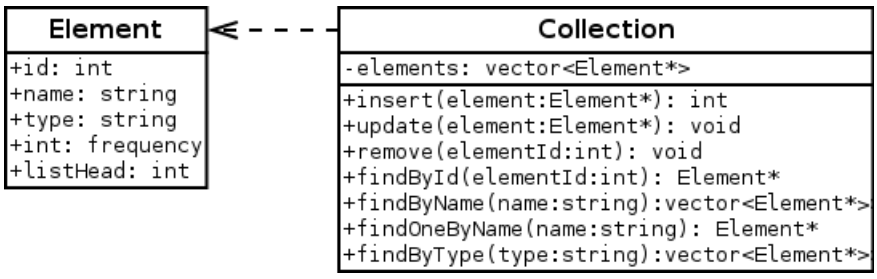
Tárolás struktúra megtervezése

A MongoDB használata jelentősen megkönnyítette a webalkalmazás elkészítését. Elég rugalmas ahhoz, hogy a címkézéssel kapcsolatos lekérdezések is egyszerűen megoldhatók legyenek. Az dokumentumok általános formátuma viszont teljesítmény szempontjából kevésbé hatékony. A bemutatott címkézési módszer csak néhány, viszonylag könnyen implementálható adatstruktúrát és algoritmust használ, ezért érdemesnek látszott C++ programozási nyelven saját adatbázismotort készíteni.

7.1. *Az adatbázismotor felépítése*

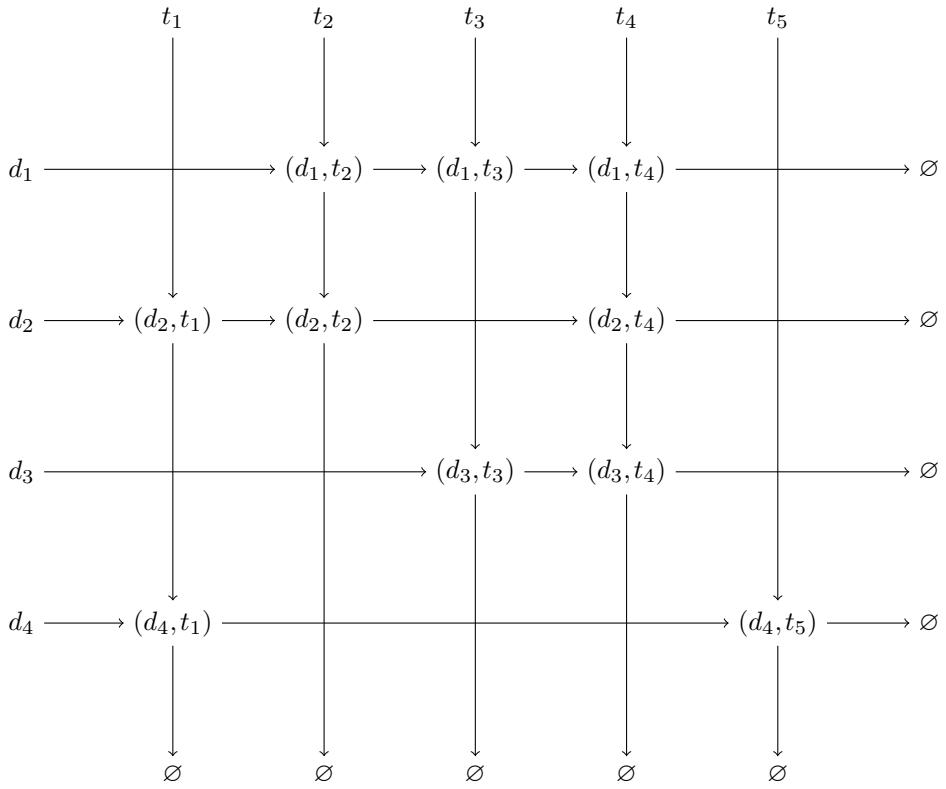
Az adatbázismotor felépítéséhez a MongoDB biztos támpontot adott. Az első lépés a kollekciók és az azokban szereplő elemek megvalósítása volt C++ osztályokként. Ezeknek az új adatbázismotorban jóval kevesebb funkció jutott.

A dokumentumoknak és címkéiknek sok közös vonásuk van. MongoDB-ben ezek adatainak tárolási formája JSON dokumentum lenne, viszont hogy a nevek ne ütközzenek itt az általános elemet az `Element` osztály reprezentálja (7.1 ábra). Ennek mindenképpen kell, hogy legyen egy egyedi azonosítója (`id`). A dokumentumoknak és a címkéknek is van egy neve (`name`), van típusuk (`type`), számon kell tartani hogy milyen gyakran fordulnak elő (`frequency`), illetve hogy a kapcsolatokat leíró adatok hol kezdődnek (`listHead`).



7.1. ábra. Az általános elemeket és az ezeket tároló kollekciókat reprezentáló osztályok.

Az elemek tárolása kollekciókban (`Collection`) történik. A két alapvető művelet az új elem beszúrása (`insert`), és az elem törlése (`remove`). A visszakereséshez a `find` nevű metódusok használhatók. Egy elemet vissza lehet keresni azonosító (`findById`), név (`findByName`) továbbá típus (`findByType`) alapján is. A címkék neveinek mindenképpen egyedieknek kell lenniük. A `findOneByName` olyan név szerinti keresést valósít meg, mely csak egy elemet ad vissza. Ez természetesen nem garantálja, hogy a címkenevek valóban egyediek is lesznek, arra az elem beszúrásánál külön ügyelni kell.



7.2. ábra. A ritka mátrix láncolt tárolási formájának egy sematikus ábrája.

A név szerinti kereséshez az egyik legjobb adatstruktúra a *trie*. Ez a kereséshez használt név hosszának függvényében lineáris időben képes megtalálni a hozzá tartozó azonosítót, vagy jelezni, hogy a név hiányzik. A kollekciók számára ez azért is előnyös, mert ebből a nevek rendezett sorrendben is visszanyerhetők.

A dokumentumok és a címkék közötti kapcsolatok tárolásához az adatstruktúrát olyan láncolt listák adják, melyeknek közösek az elemeik a hozzárendelési pontokban (7.2 ábra). Külön, egymástól független láncolt listák is elegendőek lennének, viszont így egyszerűbb ellenőrízni az adatok konzisztenciáját. A dokumentumlistákon végighaladva a csomópontokban megtalálhatóak a dokumentum azonosítói, ami itt ugyan redundanciának tünhet, viszont a címkék listájában pont ez jelzi, hogy melyik dokumentum tartozik hozzá. Ez természetesen a címkék listájára is fennáll, vagyis az elemeken haladva mindig ugyanannak a címkeazonosítónak kell következnie.

Egy csomópontnak (`Relation`) az alábbi információkat kell tartalmaznia:

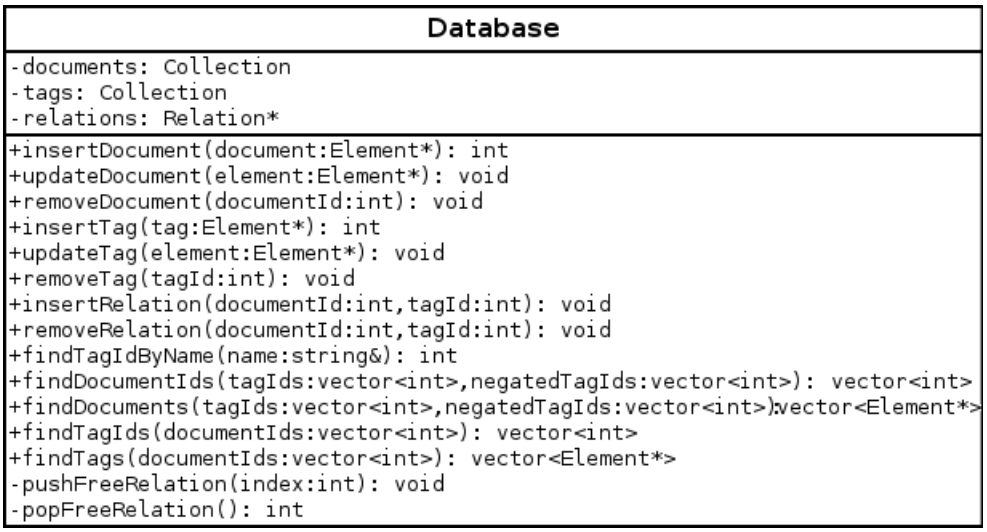
- `documentId`: A sornak megfelelő dokumentumnak az azonosítója.
- `tagId`: Az oszlopnak megfelelő címke azonosítója.
- `nextDocument`: Hivatkozás a következő, azonos `tagId`-val rendelkező csomópontra.
- `nextTag`: Hivatkozás a következő azonos `documentId`-val rendelkező csomópontra.

A ritka mátrix tárolási módja felépíthető a C++ allokációs és mutatókezelési módjával is, viszont hatékonyabb implementációt eredményez az indexeléssel való megoldása.

Mindegyik azonosító jelen implementációban 4 bájtos egészként tárolandó, így tehát minden csomópont 16 bájtot foglal. Ezeknek előre lefoglalható egy nagyobb terület, melyet aztán már hézag nélkül ki lehet tölteni. Ennek egyik előnye az, hogy az indexelésből eredően gyorsabban fog futni, illetve az adatok lemezre mentése és lemezről beolvasása

is egy lépésben megoldható, mivel nem kell külön újraszámolni a csomópontok memóriacímeit. A visszafelé láncolás sok előnnyel járna, viszont akkor egy csomópont már 24 bájtot foglalna, ami másfélszeresére növelné a szükséges tömb méretét. Az adott algoritmus igényeinek megfelelően ezekből visszafelé láncolt listák is felépíthetők, ha erre szükség van.

Az adatbázis feladata a dokumentumok és címkék kollekcióinak, valamint az ezek elemei közötti kapcsolatok kezelése, egységes felület biztosítása a lekérdezéshez. Az adatbázis osztályában (7.3) jól elkülönülnek az adatok módosítására (felvitelükre, átírásukra, törlésükre) és a lekérdezésükre szolgáló függvények.



7.3. ábra. Az adatbázist reprezentáló osztály.

Az elemek felvitelénél a megfelelő kollekcióba való beszúráson túl nincs egyéb tennivaló. A dokumentum és címke közötti kapcsolat felvitelénél növelni kell az adott elemek gyakoriság attribútumát, kapcsolat törlésénél pedig csökkenteni. Egy elem törléséhez törölni kell először minden kapcsolatát, majd csak az után lehet eltávolítani a kollekcióból is. A felviteli és módosítási műveletek esetében a címkenevek egyediségének megtartása igényel külön figyelmet.

A reláció törlésénél a `relations` tömbben lyukak keletkezhetnek. Az így felszabaduló területek használatbavételének egyik módja, hogy nyilvántartjuk ezeket a területeket, és a következő elem lefoglalásánál ezekből használunk fel. Az lényegtelen, hogy éppen melyiket is ezek közül, ezért egy láncolt lista megfelelő, melynek mindig az elejére szúrjuk be a szabad elemet (`pushFreeRelation`), vagy onnan vesszük le (`popFreeRelation`). Mivel a `relation` tömb egyébként is láncolt listákat tárol, ezért ez a lista is tárolható benne. A fej elem a tömb eleme lesz. A megoldás előnye, hogy gyors, mivel a törlés konstans időben elvégezhető, érzékeny viszont arra az esetre, ha a beszúráshoz képest jóval több törlési művelet van.

Egy másik szóbajöhető megoldás, hogy visszakeressük az utolsó elem helyét a láncolt listákban. A csomópont tartalmazza, a dokumentum és címke azonosítóját is, így a kollekciókból visszakereshető, hogy mely két lista tartozik hozzá. Ez után meg kell keresni a csomópont előtti elemeket, és így a megfelelő indexek átírásával át lehet helyezni a tömb utolsó elemét a törölt helyére. A tömb így biztosan kitöltött marad, viszont a törlési művelet lassabb lesz.

7.2. *Keresési műveletek*

Mind a dokumentumok, mind pedig a címkék azonosítói értelemszerűen egyediek, illetve értelmezhető rajtuk rendezési reláció. Új elemet beszúrni az ismertetett láncolt listás szerkezetbe több helyre is lehetne. A beszúrás szempontjából a leghatékonyabb megoldás az, ha ez a lista elejére, konstans műveleti idővel történik. A keresés viszont így bonyolultabb, mivel nem lehetne tudni, hogy egy adott azonosítón túlhaladtunk-e már. Az azonosítók rendezettségének megtartásához minden beszúrásnál meg kell keresni az új elem helyét a listákban, ami időt vesz igénybe ugyan, de a keresési művelet jóval gyorsabb lesz.

Az egyszerűség kedvéért érdemes azt az esetet vizsgálni először, amikor negált címkék nélkül keresünk dokumentumokat. A feladat úgy fogalmazható át, hogy a kiválasztott oszlopokban keressük azokat a sorokat, melyeknél mindenütt van csomópont.

Jelölje itt a `tags` a keresésben szereplő címkéket, az `nTags` pedig ezek számát. A `documents` tömb tárolja a címkékhez tartozó listák csomópontjainak `documentId` értékét. A `relations` tömb itt is a kapcsolatokat leíró listákat tároló tömb, a `next` pedig az ebben, a megfelelő lista szerinti következő csomópont indexe. A keresési algoritmus ezeket felhasználva az alábbiak szerint alakul.

```
for i = 1 to nTags
    next[i] = tags[i].listHead

    if next[i] == 0
        return

    documents[i] = relations[next[i]].documentId
    next[i] = relations[next[i]].nextDocument

while true
    i = 1
    while i < nTags
        if documents[i] == documents[i + 1]
            i = i + 1
        else
            if documents[i + 1] < documents[i]
                i = i + 1

            if next[i] == 0
                return

            documents[i] = relations[next[i]].documentId
            next[i] = relations[next[i]].nextDocument

            i = 1

    print documents[1]

    if next[1] == 0
        return
    else
```

```
        documents[1] = relations[next[1]].documentId
        next[1] = relations[next[1]].nextDocument
```

A címkékre történő keresés teljesen analóg módon működik. A negált címkék egy plusz szűrési feltételt adnak. Elegendő csak akkor figyelembe venni őket, ha a többi címke már egyezik. Vezessük be a `negatedTags`, `negatedDocuments` és `negatedNext` változókat, az elemszámukat pedig jelölje `nNegatedTags`. Az új tömbök inicializálása lényegében ugyanaz mint a nem negáltaké. A szűréshez az előbbi kódrészben a `print` sor a következővel helyettesítendő.

```
filtered = false
for i = 1 to nNegatedTags
    while negatedDocuments[i] != 0 and
          negatedDocuments[i] < documents[1]
        if negatedNext[i] != 0

            negatedDocuments[i]
              = relations[negatedNext[i]].documentId

            negatedNext[i]
              = relations[negatedNext[i]].nextDocument
        else
            negatedDocuments[i] = 0

    if negatedDocuments[i] == documents[1]
        filtered = true
        break

if filtered == false
    print documents[1]
```

Az összefűzött láncolt listás szerkezetnek köszönhetően lehetőség van a keresés optimalizálására. Olyan esetben például, ha egy gyakori címke is szerepel a keresésben, akkor érdemes a legrövidebb címkelistán haladni, majd az egyes dokumentumokon nézni végig, hogy szerepelnek-e rajta az adott címkék. Tehát a két keresési irány között rugalmasan lehet váltani.

7.3. *A keresés hatékonysága*

Az adatbázis belső szerkezete kialakításából adódóan támogatja az összetett, több címkéből álló kereséseket is. A teljesítményét más adatbázismotorokéval összehasonlítani csak úgy lehet, ha azonos funkcionalitást implementáljuk azokban is. Az így kapott eredmények is félrevezetőek lehetnek, mivel a keresési hatékonyság mellett rengeteg más szempontot is figyelembe kellene venni.

A hierarchiában való navigálással is összevethető. Ekkor a korrekt összevetéshez az azonos keresési feltételek megteremtésére kell nagyobb figyelmet fordítani. A hierarchiában való, csomópontok sorozatán történő navigáláshoz képest viszont biztosan költségesebbek ezek a műveletek. Ez a megoldás nyilván a szerkezetnek megfelelő lekérdezések esetén lesz hatékonyabb. Az egyik feltételezés az, hogy a felhasználói szokások

már változtak annyit az utóbbi években, hogy ilyen jellegű keresésekre már gyakrabban szükség lehet, így összességében jobb válaszidők érhetők el ezzel a megoldással.

A keresés hatékonyságát természetesen indexek használatával itt is lehet javítani. Azt, hogy mit és hogyan érdemes indexelni majd csak nagyobb mennyiségű használati statisztika alapján mondható meg.

7.4. *További adatbázis funkciók*

Ahhoz, hogy adatbázismotorról beszélhessünk még számos további funkcióra szükség van. Ilyen alapvető dolgok például, hogy az adatok perzisztensen tárolásra kerüljenek, a műveleteket naplózza a rendszer vagy hogy a párhuzamos hozzáférés is meg legyen oldva. Jelenleg csak azok az összetevők készültek el, melyek a webalkalmazás esetén is létfontosságúak voltak.

8. fejezet

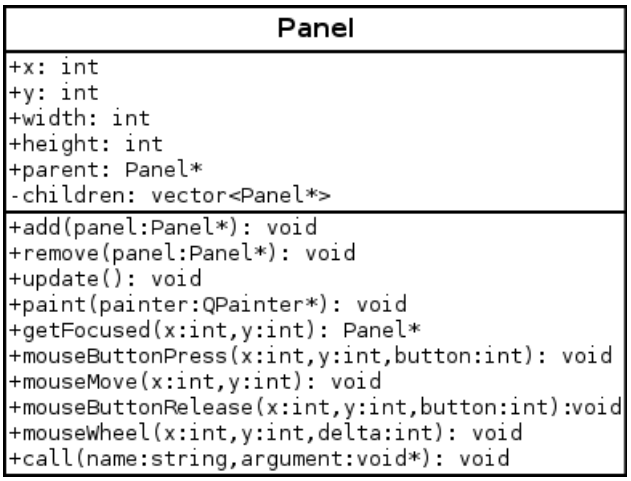
Fájlkezelő alkalmazás

A dolgozatban a címkézés elsődlegesen mint a felhasználó dokumentumainak rendszerezését szolgáló eszköz szerepel. Ehhez mindenképpen szükség van egy olyan fájlkezelő alkalmazásra is, mely ezt lehetővé teszi. A következőkben ennek egy lehetséges C++/Qt-s megvalósítási módjáról lesz szó.

8.1. *A felület szerkezete és működése*

A Qt [KD02] számos felhasználói felület elemmel (*widget*) rendelkezik, viszont ebben az esetben érdemesnek látszott újakat is létrehozni. Egyik ok az egységes grafikus megjelenítés volt, a másik pedig, hogy így a program más megjelenítő környezetbe (például GTK+[HP99], vagy Java Swing [RE98]) egyszerűbben átvihető lesz. Ehhez a megjelenítő, egér- és billentyűzetesemény-kezelő részeket kell átírni, az alkalmazás belső működése nem igényel változtatást.

A kialakított felület alapvető eleme a `Panel` (8.1). Ez egy téglalap alakú területet kezel. A panelekhez az `add` metódusukkal paneleket lehet hozzáadni, a `remove`-val pedig el lehet távolítani azokat. A `children` vektor tárolja az ezekre mutató pointereket.



8.1. ábra. A `Panel` osztály felépítése.

A panel `paint` metódusa rajzolja ki magát a felületi elemet. A teljes felület kirajzolása rekurzívan történik; minden panel meghívja a gyerekpanelek rajzoló metódusait.

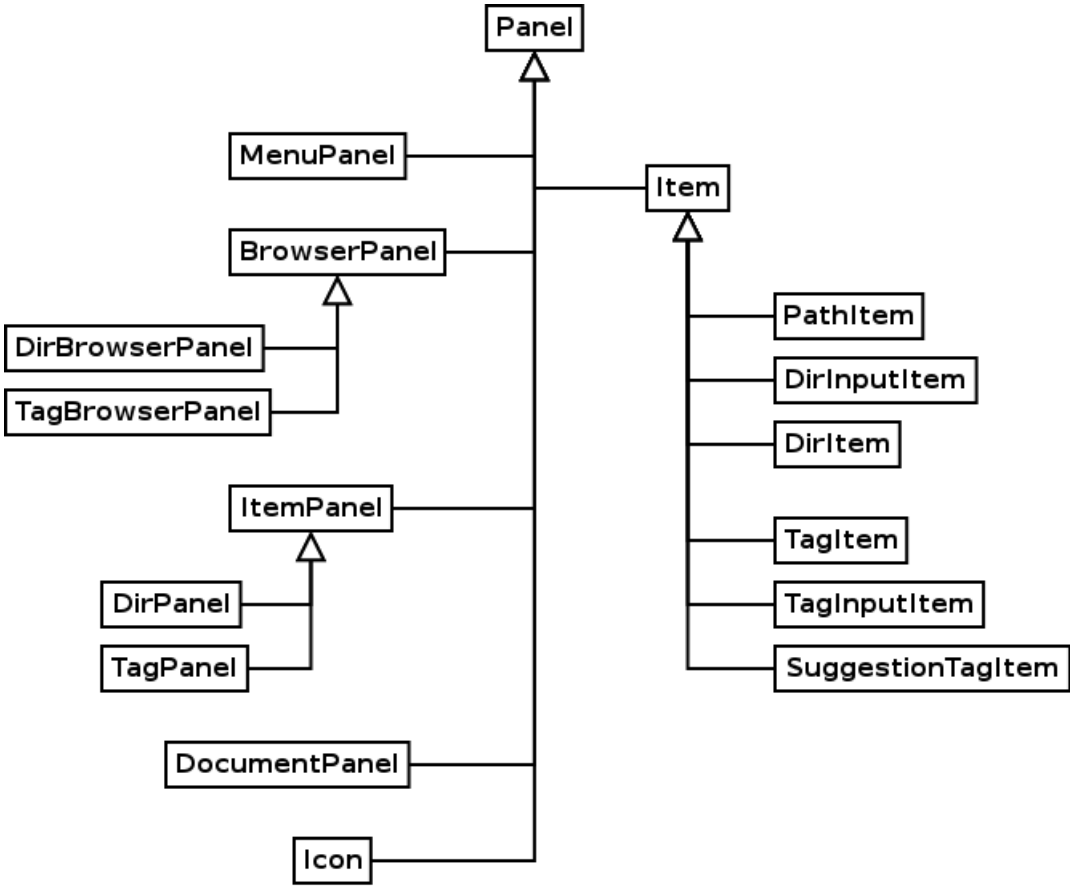
A panelen belül az elrendezésért az `update` metódus felelős. Ez adja meg a gyerekelemek pozícióját és méretét, majd a rajzoláshoz hasonlóan rekurzív hívások következnek

a panel-hierarchiában lefelé.

Az egéreseményeket kezelését a `mouse` kezdetű metódusok végzik. Az egéreseményt magát először a Qt-s eseménykezelő fogadja. Ahhoz, hogy ez egy adott, a hierarchiában lejjebb lévő panelhez eljusson először a gyökérpanel `getFocused` metódusával kell lekérdezni, hogy éppen melyik panel van az adott pozíción. Ez szintén úgy történik, hogy a gyökérpanel meghívja sorban a gyerekpanelek `getFocused` metódusait, és ha azok 0 értékkel térnek vissza, akkor a panel visszatér a saját referenciájával. Miután megvan az a panel, amelyre az esemény vonatkozik már közvetlenül meg lehet hívni az adott eseményhez tartozó metódusát.

A panelhierarchiában lefelé a `children` vektor alapján tarthatják egymással a kapcsolatot az elemek. Azért, hogy ez felfelé is működjön minden panel a `parent` adattagjában tárolja, hogy melyik szülőpanelhez tartozik. Az üzenetváltáshoz a `call` függvény használható. Az üzenet küldője a `name` argumentumban meg kell hogy adja, hogy mi a hívás oka, az `argument`-ben pedig az üzenethez tartozó esetleges további paramétereket.

A `Panel`-ből származó további osztályok a 8.2 ábrán láthatóak. Mint arról már korábban szó volt (2.1) általában a kétpaneles kialakítás a szerencsés. Mindkét panel `BrowserPanel` típusú objektum. Attól függően, hogy a hagyományosabb hierarchikus, vagy a címkézés alapú kezelésre van-e szükség ez lehet `DirBrowserPanel` vagy `TagBrowserPanel` is.



8.2. ábra. A felhasználói felület elemeit reprezentáló osztályok hierarchiája.

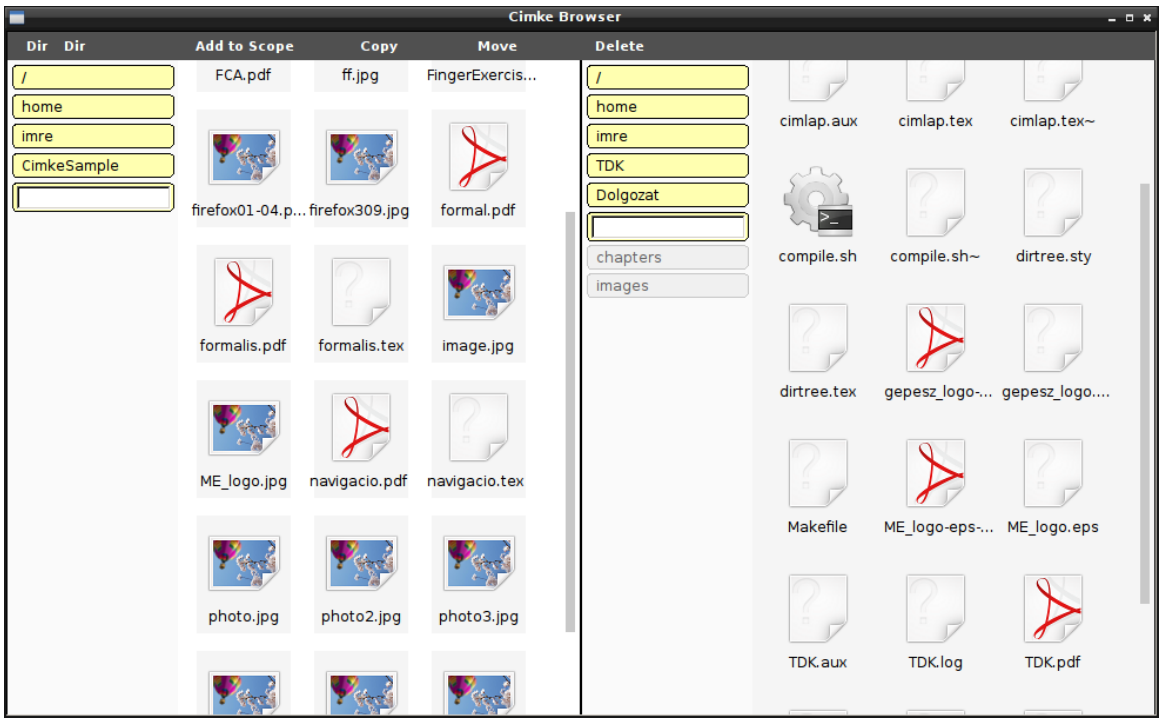
Az `ItemPanel` általános listakezelési feladatokat lát el. Ennek is két altípusa lehet; `DirPanel` a jegyzékek kezeléséhez és `TagPanel` a címkékhez. Ez, és egy `DocumentPanel` tartozik hozzá a `BrowserPanel`-hez. A `DocumentPanel` csak a megjelenítendő dokumentumokhoz tartozó adatokat kapja meg, majd feltölti a `children` vektorát az ez alapján létrehozott `Icon` típusú objektumokkal.

A listákban `Item` típusú objektumok szerepelnek. Ezek altípusai követik az oldalsáv hármas tagozódását. Az elérési úthoz vagy keresésben szereplő címkékhez a `PathItem` vagy `TagItem`, a szöveges bemenet kezeléséhez a `DirInputItem` vagy `TagInputItem`, a jegyzékek vagy várható címkék listájához pedig a `DirItem` vagy `SuggestionItem` típusúak tartozhatnak, a szülő panel típusának megfelelően.

A megjelenítéshez ugyan jól használható ez a fajta, panel alapú felépítés, viszont a szöveges bemenethez a Qt saját `QLineEditor` objektuma kellett. A szövegbevitelhez a `DirInputItem` vagy `TagInputItem` mindenképpen a panelhierarchia alján kap helyet, de innen önállóan nem tud létrehozni új widget-et. Szükséges volt még egy `TextInputManager` segédosztály is. Ez az alkalmazás indulásakor kap néhány `QLineEdit` objektumot, melyet a `getTextInput` metódusán keresztül át tud adni az adott panelnek, majd a `releaseTextInput`-tal azt vissza is tudja venni.

Egy további segédosztály még a `PreviewManager`, ami az előnézeteket kezeli. A gombok és ikonok megjelenítésekor is gyakran kell ugyanazokat a képeket megjeleníteni. Azokat nyilván nem érdemes minden alkalommal megnyitni, ezért ez az osztály előre betölti őket. A `getTagButton` metódusával a gombok, a `getPixmap`-pel pedig az ikonok képeit adja vissza.

Az alkalmazás továbbra is fejlesztés alatt áll. A dolgozat írásakor aktuális állapotáról látható egy kép a 8.3 ábrán. Ez az eddig vázolt szerkezetnek és működésnek megfelelően készült, viszont rengeteg, más fájlkezelő alkalmazásokban természetesnek mondható művelet még hiányzik belőle. Ezek elkészítése inkább csak technikai, nem pedig elvi kérdéseket vet fel.



8.3. ábra. Kép az elkészült alkalmazásról.

8.2. *Kapcsolat az adatbázissal és a fájlrendszerrel*

Az adatbázissal való összekapcsolás több módon is elképzelhető.

- Az adatbázist függvénykönyvtárként elkészítve azt az alkalmazással együtt le lehet fordítani. Előnye, hogy egyszerűen kivitelezhető, és a fájlkezelő programnak így nem lesznek utána külső függőségei. Hátrány viszont, hogy több futó példány esetén azok működését össze kell hangolni az inkonzisztens állapotok elkerülésére.
- Az adatbázis működhet, mint operációsrendszeri szolgáltatás is. Ez jobb megoldás, mivel több alkalmazás is közösen használhatja. Viszont a kapcsolatkezelést, védelmi funkciókat és operációsrendszerre való telepítést is meg kell oldani.

Jelenleg az adatbázis forráskódja egy `db` jegyzékben szerepel a fájlkezelő forráskódjában. A fejlesztés során, így az adatbázist is lehetett módosítani, és az összekapcsolás is adott volt. A későbbiekre nézve az előbbi két pont fejlesztési célként szerepel.

A címkézett dokumentumok a fájlrendszerben egy helyen, névként csak az adatbázisbeli azonosítójukkal ellátva szerepelnek. Ahhoz, hogy ezt más alkalmazások is probléma nélkül tudják használni az eredeti nevükkel egy linket kell létrehozni, és ezt kell átadni majd. A használaton kívüli linkeket később törölni lehet. Arra is kell figyelni, ha az alkalmazás például ideiglenes fájlokat hozna létre. Ezt jelezni kell a felhasználónak, vagy pedig automatikusan felvenni ezeket a fájlokat is az adatbázisba az eredeti fájlok mellé (az eredeti fájléval megegyező címkéket adva hozzá).

8.3. *Kiegészítő funkciók*

Ahhoz, hogy az alkalmazást napi szinten használni lehessen további elemző, és kényelmi funkciókra is szükség van. Ilyenek például az alábbiak.

- *Adatbázis statisztikák*: Meg kell tudni néznie a felhasználónak, hogy mennyi helyet foglalnak az adatbázisban lévő dokumentumok, mennyi van belőlük, milyen címkék vannak, melyek a gyakrabban használtak. Új adatbázist beolvasva segítheti a tájékozódást, ha az adatokból kinyert fogalmi hálót is meg lehet tekinteni.
- *Előzmények*: Az összes fájlműveletet naplóznia kell a programnak. Vissza kell tudni keresni, hogy melyik dokumentumon, mikor és milyen változtatás történt, esetleg néhány visszavonására is lehetőséget kell adni.
- *Lomtár*: A törlendő dokumentumokat (ha erre van igény a felhasználó részéről) ideiglenesen tudni kell tárolni. Ezeket vissza is kell tudni állítani az eredeti állapotukba, ha mégsem kerülnének törlésre.

Az fájlkezelőnek testreszabhatónak kell lennie. Ebbe beletartoznak a nem funkcionális beállítások is, mint például hogy milyen színek, betűtípusok, ikonok legyenek. Ezek szintén nem újszerű dolgok, viszont a fejlesztésnél tekintettel kell lenni ezekre a szempontokra is.

A fejlesztés még nagyon korai stádiumában van. Az eredeti fájlrendszer böngészésére már az új felületen keresztül van mód. A címkézés felhasználói felület része még nem tekinthető megoldottnak, illetve az alapvető fájlműveletek is hiányoznak. Ezek elkészítése itt sem elvi, inkább csak technikai problémákat vethet fel.

A program elsődlegesen Linux operációs rendszerhez készült. A Qt-nak köszönhetően más rendszerekre könnyen átvihető. A fájlok és eszközök kezelése igényelhet különösebb figyelmet.

9. fejezet

További alkalmazási lehetőségek

9.1. *Címkézés alapú fájlrendszer*

A címkézéses megoldást elsősorban a felhasználók saját dokumentumainak nyilvántartásához szokták használni. A megvalósítása is ennek megfelelően úgy történik, hogy a meglévő hierarchikus fájlrendszerre építenek még egy absztrakciós réteget, amely kezeli a címkék és az elérési útvonalak közötti megfeleltetést. Ezzel a meglévő rendszer funkcionalitását bővítik úgy, hogy nem változtatják meg az addigi tárolási koncepciót. Vizsgálandó, hogy mi is az igazi oka annak, hogy még nem készült és terjedt el olyan fájlrendszer, amely kizárólagosan címkézést használna.

Az egyik ok az lehet, hogy egy egész fájlrendszer kialakítása túlságosan bonyolult feladat az eddig felmerült igényekhez képest. Tovább nehezíti ezt, hogy az operációs rendszerek hierarchikus fájlrendszerekre épülnek, tehát egy ilyen változtatás a kernel, esetlegesen a hozzá tartozó függvénykönyvtárak egy részének, illetve nem utolsó sorban a rendszer- és felhasználói programoknak az átírását vonná maga után.

Az okok között szerepelhet továbbá, hogy rendszer szinten a hierarchia megfelelő mind szemantikailag mind pedig teljesítményt illetően. A UNIX fájlrendszerekben minden fájl, melyek egyetlen nagy fában helyezkednek el [RR04]. Közelebbről szemügyre véve a jegyzékek neveit felvetődhet néhány probléma. Például mi a viszonya a `/bin` és a `/sbin` jegyzéknek ? Az elnevezések rövidségét részben az indokolja, hogy könnyen be lehessen őket gépelni, viszont a csoportosítás egy beszédesebb elnevezés esetében sem lenne egyszerűbb. Mindkét jegyzék binárisokat tartalmaz, tehát valahogyan együtt kellene kezelni őket. Binárisokkal többek között még az `/usr` jegyzéken belül is találkozhatunk, ahol szintén van egy `/bin` és `/sbin` jegyzék.

A `lib` nevű jegyzékekből is több van, például `/lib`, `/usr/lib`, `/var/lib` esetleg még `/usr/local/lib` is található a rendszerek egy részén. A `/lib`, `/lib/usr`, `/lib/var`, `/lib/usr/local` kialakítás is megfelelő lenne?

A rendszer ilyesfajta kialakításának nyilvánvalóan komoly okai vannak. Ez a pár példa csak arra próbálta felhívni a figyelmet, hogy még egy ennyire kiforrott, széles körben elterjedt rendszer esetén is könnyen lehet találni olyan kényszermegoldásokat, amelyek láthatóan a hierarchia adta korlátoknak tudhatók be.

Voltak már kísérletek a UNIX fájlrendszer átalakítására vonatkozóan. Az egyik radikális lépést ezek közül a GoboLinux[1] tette meg. Ez egy forráskód alapú disztribúció, amely a programokat, mint egységet igyekezett megtartani. Minden program, értve ezalatt a függvénykönyvtárakat is a `/Programs` jegyzékben kapott helyet. A programok telepítése és törlése így lényegesen átláthatóbbá vált, csak a linkeket kellett beállítani a futtatható-, konfigurációs állományoknak és függvénykönyvtáraknak. További eltérések

[1] [GoboLinux] http://www.gobolinux.org

is voltak a hagyományos UNIX szerkezethez képest. Viszont ennél is csak egy kernelbeli módosítás volt az, amely a háttérben leképezte az újfajta elérési útvonalakat a régire.

Tekintsünk egy programot a rendszerben, és kövessük végig a telepítését egy hierarchikus és egy címkézett rendszerben is.

Feltételezzük, hogy a program forráskód formájában állt rendelkezésre. Azt lefordítottuk és a telepítés során pedig a fájlokat a megfelelő jegyzékekbe bemásoltuk. A többi programmal is hasonlóképpen történt, így ha például a naplózás érdekel bennünket, akkor elegendő a naplófájlok jegyzékét végignézni, és ott megtalálhatjuk a többi program naplóját is. Hogyan tudjuk most megnézni, az adott programokhoz tartozó összes aktuális állományt?

Amennyiben tudjuk, hogy hogyan zajlott a telepítés, akkor ez a fájlokat külön megkeresve, vagy egy erre a célra készített segédprogrammal megoldható, viszont a fájlrendszer ebben nem sok segítséget nyújt. Nézzük meg most, hogy hogyan lehetne ezt címkézéssel megoldani.

Tegyük fel, hogy a forráskódhoz tartozó állományok először csak a program nevét (pl.: "`MintaProgram`"), azt hogy forráskódról van szó (pl.: "`source code`"), illetve néhány, a fájlokra vonatkozó speciális címkét tartalmaznak (pl.: "`header`", "`Makefile`"). A fordítást követően lássuk el a fájlokat a funkciójuknak megfelelő címkékkel (pl.: "`binary`", "`shared object`", "`log`"). Mivel az előbbiekben is telepíteni kellett a programot, ezért most adjunk hozzá egy olyan címkét ami ezt jelzi (pl.: "`__system__`"). A rendszerbinárisokat így már egy helyen meg lehet találni (pl.: `["__system__", "binary"]`), a naplófájlokat ugyancsak (pl.: `["__system__", "log"]`), viszont így már a program saját fájljai is hasonló módon elérhetővé váltak (pl.: `["MintaProgram"]`).

9.2. *Navigáció és fájlműveletek*

Nézzük meg, hogy parancssoros kialakítás esetén hogyan nézne ki a navigáció a fájlrendszerben. A gyökér jegyzéknek az üres lekérdezés felel meg. Ennek az összes dokumentumot kell visszaadnia, mivel az üres címkehalmaz mindegyiknek a részét képezi. A listázáshoz itt is használható például egy `ls` parancs.

```
[]: ls

    C.pdf
    C_programming.pdf
    C_eloadas.ppt
    kep1.png
    kep2.png
    Pointerkezeles.pdf
    Printf.pdf C-Tut-4.02.doc
```

A szögletes zárójelek itt most jelöljék a keresésben szereplő címkék halmazát. A mintarendszerben 7 dokumentum van. A címke kiválasztásához legyen egy `tag` parancs, mely a következőképpen használható.

- tag-ek kiíratása: `[]: tag filename.ext`
- tag hozzáadása: `[]: tag filename.ext +"új cimke"`
- tag eltávolítása: `[]: tag filename.ext -"törlendő cimke"`
- lekérdeshez hozzáadás (*add to set*): `[]: tag +"új cimke"`

- lekérdezésből eltávolítás (*remove from set*): `[]: tag -"törlendő cimke"`

A negált címkék jelölésére a `!` használható, például `!"doc"`.

A felsoroltak a 3.2 szakaszban bemutatott elemi műveletek megfelelői. Ezek segítségével vizsgáljuk meg pár példán keresztül, hogy hogyan is nézhetne ki a parancssor használata a gyakorlatban. Nézzük meg először, hogy az első dokumentumon milyen címkék vannak.

```
[]: tag C.pdf

    "pdf", "C", "Kernighan", "Ritchie", "book", "programming"
```

Ezek alapján adjuk hozzá a kereséshez a `pdf` címkét, majd listázzuk ki az eredményt.

```
[]: tag +"pdf"
["pdf"]: ls

    C_programming.pdf
    Pointerkezeles.pdf
```

Töltsünk most le közéjük egy újabb dokumentumot a `wget` programmal.

```
["pdf"]: wget http://www.pdfdocuments.download/Book.pdf
["pdf"]: ls

    Book.pdf
    C_programming.pdf
    Pointerkezeles.pdf
```

Adjuk ehhez is hozzá a `book` címkét, majd szűkítsük is vele a keresést.

```
["pdf"]: tag Book.pdf +"book"
["pdf"]: tag +"book"
["pdf", "book"]: ls

    Book.pdf
    C_programming.pdf
```

Tegyük fel, hogy a `Book.pdf`-re a későbbiekben mégsem lesz szükségünk, ezért töröljük le.

```
["pdf", "book"]: rm Book.pdf
["pdf", "book"]: ls

    C_programming.pdf
```

Látható, hogy a műveletek itt sem sokban térnek el a megszokottaktól.

A címke nélküli dokumentumok kezelése külön figyelmet igényel, mivel az eddigi keresési módokkal csak az üres keresés adná vissza azokat. Ehhez bevezethető például az `ls`-nek egy `-e` kapcsoló, amivel az aktuális kereséstől függetlenül lehet listázni őket.

```
["pdf", "book"]: ls -e

    kep2.png
```

A másik lehetséges módszer, hogy minden dokumentum először kap egy olyan címkét, amely jelzi, hogy nincs még rajta kereséshez használható címke. Az első címke hozzáadásakor ezt törölni kellene róla. Amennyiben ez például az üres nevű címke, akkor az összes címkézetlen dokumentum lekérdezése a következők szerint alakul.

```
["pdf", "book"]: ls [""]
```

A szögletes zárójelekkel megadott címkehalmaz az aktuális hatáskör abszolút elérési "útvonalaként" kezelhető. A címkékre vonatkozó navigációs műveletek megadásával relatívak is képezhetők, melyek összefoghatják az elemi műveleteket. A következő példa egy áthelyezést mutat be ezek használatával:

```
["pdf", "book", "C", "prog"]: mv Java.pdf -"C", +"Java"
```

Az előbbi átnevezéssel is kiegészítve:

```
["pdf", "book", "C", "prog"]: mv Java.pdf -"C", +"Java": J.pdf
```

A `+` és `-` itt az aktuális címkelistához képest adják meg azokat a változtatásokat, melyeket elvégezve kapjuk a célcímet. Az abszolút helyre az áthelyezés az eddigiek alapján ugyancsak adódik:

```
["pdf", "book", "C", "prog"]: mv image.png ["images"]
```

A bemutatott néhány új jelölés és parancs bevezetése tehát elegendő az alapvető fájlműveletek elvégzéséhez és a kereséshez. Különböző egyéb jelölési módok is megadhatók. A példák elsődlegesen azt szolgálták, hogy valószerű eseteken keresztül vizsgálhassuk meg a címkézés módszerét.

9.3. *Felhasználók és jogosultságaik kezelése*

A fájlrendszerekben a jogosultságokat is kezelni kell. A címkézéssel ez, és a felhasználók csoportokba rendezése tulajdonképpen egyszerre megoldható.

A program telepítésére hozott példában a `__system__` címke jelölheti a fájlok tulajdonosát is, ahol tehát ez maga az operációs rendszer. Ezekhez a hozzáférést, vagy akárcsak a listázást általában nem célszerű lehetővé tenni a felhasználói jogkörökben.

Az állományokhoz való hozzáférés kezelésének több módja is lehet. Kedvezőbbnek tünik az a megoldás, ha a felhasználóhoz, vagy csoporthoz tartozó címkék szerepelnek minden lekérdezésben, de azok nem látszódnak, és nem is lehet őket a felhasználói jogkörben eltávolítani.

A címkézés lehetővé teszi továbbá azt is, hogy azonos nevű fájlokat úgy helyezzünk el, hogy csak a felhasználót azonosító címke tegye azt egyértelművé. A rendszer karbantartása szempontjából ez szintén könnyebbséget jelenthet. A hierarchia bizonyos pontjairól összegyűjtendő fájlok itt természetes módon egy helyen jelenhetnek meg.

10. fejezet

Összegzés

Összegezve az eddigi tapasztalatokat a címkézésnek számos előnye van. A felhasználótól szemléletváltást, új fogalmak megtanulását, megértését igényli, viszont ezek nem bonyolultabbak a jelenlegi rendszerekben használatosaknál. Ez az új felhasználói felület esetében is igaz, tehát egy kis időbe telik megszokni, de az újszerű rendszerek többségénél ez így van.

A fogalomhálók elemzésével láthatóvá vált a hierarchikus módszer és címkézett megközelítés közötti kapcsolat. A két, látszólag különböző rendszerezési mód elemeit megfeleltetve egymással a hierarchikus nyilvántartások, mint a címkézettek speciális esetei jelentek meg.

A címkézés, a jelenleg elterjedt alkalmazási területeken túl egy általánosabb rendszerezési eszközként is használható. A dolgozatban ennek alapelvei példákkal illusztrálva szerepelnek, illetve a bemutatott programokon keresztül végig lehet követni egy ilyen elveken alapuló rendszer fejlesztésének a menetét. Természetesen ahhoz, hogy ezekből a felhasználók számára elérhető, a mindennapokban jól használható alkalmazások legyenek még további fejlesztésekre van szükség.

A dolgozatban leírtam az alapelveket, és részleteztem az eddigiekben felmerült fejlesztési lehetőségeket is. A hiányzó részeket természetesen a továbbiakban kifejlesztem.

Irodalomjegyzék

[AS09] Andrews, S.: *In-Close, a fast algorithm for computing formal concepts*, International Conference on Conceptual Structures (ICCS), Moscow, 2009.

[BG06] Bernhard Ganter: *Finger Excercises in Formal Concept Analysis*, Dresden ICCL Summer School, 2006.

[BG84] Bernhard Ganter: *Two basic algorithms in concept analysis*, FB4-Preprint Nr. 831, 1984.

[DOM] World Wide Web Consortium: *Document Object Model*, `http://www.w3.org/DOM`, Internet, 2012.

[Elyse] Silkwood Software: *Elyse, freedom from folders*, `http://silkwoodsoftware.com` Internet, 2012.

[GoboLinux] GoboLinux, Linux disztribúció weboldala, `http://www.gobolinux.org` Internet, 2012.

[GW05] Bernhard Ganter, Gerd Stumme, Rudolf Wille: *Formal Concept Analysis, Foundations and Applications*, Springer, 2005.

[HP99] Havoc Pennington: *GTK+/Gnome Application Developement* New Riders Publishing, 1999.

[JSON] *JavaScript Object Notation*, `http://www.json.org` Internet, 2012.

[KC10] Kristina Chodorow, Michael Dirolf: *MongoDB: The Definitive Guide*, O'Reilly Media, 2012.

[KD02] Matthias Kalle Dalheimer: *Programming with Qt, Writing Portable GUI applications on Unix and Win32*, O'Reilly Media, 2002.

[KL03] Kovács László: *Algorithms for Building Concept Set and Concept Lattice*, Production Systems and Information Engineering, Miskolci Egyetem, **1**, 2003, 91-106.

[KL07] Kovács László: *Generating Decision Tree from Lattice for Classification*, Proceeding of ICAI07, Eger, 2007, 377-385.

[Nepomuk] *Networked Environment for Personalized, Ontology-based Management of Unified Knowledge*, `http://nepomuk.semanticdesktop.org/nepomuk` Internet, 2012.

[OpenKM] OpenKM: *Open Knowledge Management*, `http://www.openkm.com` Internet, 2012.

[RB08] Radim Belohlávek: *Introduction to Formal Concept Analysis*, Palacky University, Olomouc, 2008.

[RE98] Robert Eckstein, Marc Loy, Dave Wood: *Java Swing*, O'Reilly Media, 1998.

[RK06] Körei, A. and Radeleczki, S.: *Box elements in concept lattices*, Contributions to ICFCA 2006, 4-th International Conference on Formal Concept Analysis, Dresden, Febr. 12-17, 2006. pp. 41-55.

[RS98] Radeleczki, S.: *A fogalomhálók egy műszaki alkalmazása*, International Conference on Computer Science, "MicroCAD'98", Section Applied Mathematics, University of Miskolc, March 12., 1998. pp. 1-7.

[RR04] Rusty Russel, Daniel Quinlan, Christopher Yeoh: *Filesystem Hierarchy Standard*, Filesystem Hierarchy Standard Group, 2004.

[SF12] Steve Francia: *MongoDB and PHP*, O'Reilly Media, 2012.

[SJ11] Stuart Jarvis: *Organize Your Life with Nepomuk* `http://www.linuxjournal.com/article/10911` Linux Journal, 2011.

[Tabbles] Yellow Blue Soft: *Tabbles, folders evolved*, `http://tabbles.net` Internet, 2012.

[Tag2find] *Tag2find, Tag everything on your desktop*, `http://www.tag2find.com` Internet, 2012.

[TaggedFrog] LunarFrog Software: *TaggedFrog*, `http://lunarfrog.com/taggedfrog` Internet, 2012.

[Tracker] Ubuntu Wiki: *Tracker* `https://wiki.ubuntu.com/Tracker` Internet, 2012.

[VL10] Dr. Veres Laura: *Osztályozási fák, durva halmazok és alkalmazásaik*, PhD értekezés, Miskolci Egyetem, 2010.

[WB04] Bastian Wormuth, Peter Becker: *Introduction to Formal Concept Analysis*, 2^{nd} Internation Conference of Formal Concept Analysis, Sydney, Australia, 2004.