Tartalomjegyzék

1. Bevezetés

1.1. A blogokról

A modern internetes világ egyik igen fontos és népszerű eleme a blog. Habár sokan új találmányként gondolnak rá, a blog, mint konkrét írásbeli kommunikációs forma lassan 15 éves múlttal rendelkezik, gyökerei pedig még mélyebbre nyúlnak vissza, még az Internet előtti időkre.

1. ábra. Tim Berners-Lee

A mai, blogoláshoz hasonló tevékenység a régi, Usenet-es időkben alakult ki, még 1983-84 környékén. Ekkor jelentek meg az első hírcsoportok, köztük a mod.ber nevű hírcsoport, ahol a tagok összefoglalókat írtak az általuk érdekesnek talált oldalakról [3].

Az első „blogbejegyzés” keletkezésének ideje máig vita tárgyát képezi. Bizonyos források[1] szerint az első naplószerű bejegyzést 1994-ben írta egy amerikai egyetemi hallgató, Justin Hall. Más források szerint az első bejegyzés Tim Berners-Lee nevéhez köthető, és „What's new in '92”[1] címmel jelent meg, 1993-ban.

A weblog kifejezés megalkotója John Barger volt, 1997-ben, a „webes napló” kifejezés (angolul web log) egybeírásával [4]. A „blog” kifejezést először Peter Merholz használta, 1999-ben, a weblog rövidítéseként [4]. Mivel ekkortájt jelentek meg az első weblogfarmok (blog szolgáltatók), a blog szócska igen hamar elterjedt a köznyelvben is.

A blog népszerűségét részben egy szomorú eseménynek is köszönheti. A 2001. szeptember 11-i terrortámadásokról, majd az ezt követő eseményekről sok civil tudósító, önkéntes újságíró írt tudósításokat, blog formátumban [3][4][6]. Ezek az összefoglalók igen nagy figyelmet kaptak, mivel az eseményekről szinte a megtörténés pillanatában lehetett értesülni.

A blog világa azonban egy kicsit később, 2002-ben mutatta meg igazi erejét. Trent Lott, akkori michigan-i szenátor a vádak szerint rasszista kijelentéseket tett, ezzel kívánt Strom Thurmond dél-karolinai szenátor előtt tisztelegni. A bloggerek folyamatosan felszínen tartották az ügyet, korábbi Lott-idézetekkel pedig bebizonyították, hogy nem egyedi, véletlen esetről van szó. Az ügy végül akkora visszhangot keltett, hogy a nyomtatott sajtóban is megjelent, igen nagy közfelháborodást keltve. Később Lott, engedve a szenátus akaratának, megvált tisztségétől [3][4][7].

1 http://www.w3.org/History/19921103-hypertext/hypertext/WWW/News/9201.html

Az ilyen, és ehhez hasonló esetek miatt az emberek szemében a blogok mára a hagyományos médiumok versenytársaivá váltak. Ez nem meglepő, hiszen a blogok naprakészek, valamint sok esetben pontosabbak, mint más médiumok, mivel a bejegyzés szerzője általában mélyebb ismeretekkel rendelkezik a témával kapcsolatban. A blogokban gyakran olyan témák is megjelennek, amelyekről más médiumok nem tudnak, (vagy talán nem akarnak) tudósítani.

Az angol Wikipedia[1] adatai szerint 2011 februárjában 156 millió nyilvános blog található a világhálón, ami elképesztő mennyiségű bejegyzést jelent. A blogoknak kereskedelmi vonzata is van, 2005-ben becslések szerint összesen 100 millió dollár értékű hirdetés ért célba a blogokon keresztül, az utóbbi évek tendenciáit figyelembe véve ez a szám tovább nőtt.

A blogok népszerűségének másik oka, hogy viszonylag egyszerű a kezelésük, nem szükséges különösebb szaktudás. Bárki, aki kedvet érez, és rendelkezik némi kitartással, valamint tehetséggel, elindíthat egy blogot, az őt legjobban érdeklő témában vagy témákban. Ma számtalan vállalkozás kínál blogszolgáltatást, többnyire teljesen ingyenesen, a használatukhoz mindössze regisztráció szükséges. Ilyen szolgáltatást a nemzetközi piacon a Google[2], a Wordpress.com[3], illetve a LiveJournal[4] kínál, míg a legismertebb magyar blogszolgáltatók a Blog.hu[5], a Blogolj.net[6] vagy a Blogter[7]. Természetesen akinek nem felel meg a nagy szolgáltatók által nyújtott blogszolgáltatás, saját tárhelyet is bérelhet, ahol egy blog kezelésére is alkalmas tartalomkezelő rendszert telepítve, elkezdheti a publikálást.

1.2. A blogok fajtái

Az idők során a blog különböző mutációkon esett át, új formátumok alakultak ki, melyek továbbfejlesztve az eredeti koncepciót, szintén igen népszerűvé váltak. Az újabb blogformátumok a következők:

- podcast: a blog szerzője egy előre felvett hangállományt tesz közzé. Ez a formátum lehetővé teszi, hogy a szerző a felvételen szintén résztvevő beszélgetőpartnerekkel is megvitassa nézeteit.
- Videoblog: a podcast-hez hasonló, itt azonban hanganyag helyett videofelvételt al-

2 http://www.blogger.com/

3 http://wordpress.com/

4 http://www.livejournal.com/

5 http://blog.hu/

6 http://blogolj.net/

7 http://www.blogter.hu/

kalmaznak

- Fotóblog: a szerző mondanivalóját képek, illetve az ezekhez írt rövid képaláírások közvetítik
- mikroblog: a második legnépszerűbb blogformátum. A szerző nagyon rövid bejegyzéseket közöl, ezek leggyakrabban csak egy linket tartalmaznak más tartalomra. A jelenleg legismertebb mikroblog szolgáltatás a Twitter, illetve a Tumblr.

Szakdolgozatomban a továbbiakban csak a hagyományos blogformátummal fogok foglalkozni.

1.3. A feladat ismertetése

Szakdolgozatom témája egy asztali blogkliens implementációs folyamatának bemutatása. Az ismertetés során kitérek a feladat meghatározására, majd azokat a technológiákat ismertetem, amelyekkel a programot meg tudom valósítani. Ezután, a tervezési fázisban a program arculatát mutatom be, majd következhet a felhasználói felület sematikus megtervezése. Ezt követően bemutatom a program egyéb alkotóelemeit, a használt osztályokat, illetve az ezeket működtető algoritmusokat.

Később ismertetem az elkészült programot, illetve megemlítek pár továbbfejlesztési lehetőséget. Végül az üzembe helyezés menetéről is szólni fogok pár szót.

Felmerülhet a kérdés, hogy mi is egy blogkliens feladata, egyáltalán mire való egy ilyen jellegű program? Mit jelent az, hogy „asztali”? Ezeket a fogalmakat szeretném most tisztázni.

Egy blogkliens feladata, hogy egy bloggal kommunikáljon, lehetőséget biztosítson a blog menedzselésére. Egy könnyen átlátható, egyszerű felületet ad a blog írójának kezébe a bejegyzések megírásához. A szerző által megírt bejegyzéseket elküldi a blogmotor számára, tárolásra, hogy az, ha szükséges, a blog olvasója számára megjeleníthesse azt. Ha valamelyik olvasó egy hozzászólással reagál a beküldött cikkre, a blogkliensben az is megjelenhet, a szerző pedig válaszolhat rá, vagy meg is változtathatja, ha a szükség úgy kívánja (a gyakorlatban ezt nevezik moderálásnak). A blogkliens felületén keresztül természetesen sok egyéb beállítást, műveletet is végrehajthatunk. A kommunikáció nyelvét, vagyis a publikációs protokollokat többen, többféleképpen meghatározták, kidolgozták, ezen protokollok egyikét, az XML-RPC protokollt a későbbiekben ismertetni fogom.

Végül is mit jelent az „asztali” kifejezés? Jelenleg kétféle blogkliens-változat létezik.

Az első változat szerint maga a blogmotor szolgáltat egy webböngészőben futó felhasználói felületet, ahol a blog feltölthető tartalommal. Az ismertebb blogmotorok és tartalomkezelő rendszerek (mint amilyen a Wordpress, a Drupal, a Serendipity, vagy a Movable Type) mind rendelkeznek ilyen felülettel.

A webes felület előnye, hogy egyszerű, látványos, ráadásul nem kell semmit telepíteni vagy módosítani a számítógépen, hiszen minden a böngészőben fut, a használata pedig nem igényel különösebb előképzettséget. Ennek következménye, hogy platformfüggetlen, gyakorlatilag bármilyen olyan operációs rendszeren, vagy hardveren használható, amely alkalmas egy webböngésző futtatására. Az egyszerűsége azonban a hátrányára is válhat, egy webes felület kevesebb dologra vehető rá, mint egy natív alkalmazás. (Ennek részben technológiai okai vannak, ezekre azonban a webes GUI keretrendszerek (mint pl. a Google Web Toolkit) megoldást jelenthetnek) Másik hátránya, hogy a használatához folyamatos internetkapcsolat szükséges, anélkül gyakorlatilag működésképtelen.

A másik szemlélet szerint a blogkliens ne a böngészőben, hanem a felhasználó számítógépén, natív alkalmazásként, a munkaasztalon (innen az „asztali” elnevezés) fusson.

Az asztali blogkliens előnye az univerzalitás: sokkal összetettebb feladatok ellátására képes, ráadásul több blog szerkesztésére is képes lehet. Lehetőség van offline cikkírásra is, ilyenkor az előzőleg megírt és publikálásra kijelölt cikkek csak akkor kerülnek beküldésre, ha rendelkezésre áll valamilyen internetkapcsolat. Hátránya viszont, hogy viszonylag bonyolultabb az üzembe helyezése, valamint több időt kell szánnunk az ismerkedésre, illetve a többletfunkciók kiismerésére.

1.4. A jelenlegi megoldások problémái

Miután eldöntöttem, hogy mi lesz a szakdolgozatom témája, feltérképeztem a feladat jelenlegi megvalósításait. Mivel az asztali blogkliensek piaca viszonylag új terület, kevés a „versenytárs”. A programok között természetesen akadt zárt, illetve nyílt forráskódú, valamint ingyenes és fizetős versenyző is. Az általam vizsgált blogkliensek leggyakoribb problémái a következők voltak:

- „régimódi” felhasználói felület: a jelenleg létező implementációk felhasználói felülete a tradicionális menü-eszköztár-munkaterület-állapotsor modellt követi. Ezzel nincs semmi baj, egy huszonpár éve alkalmazott, jól bevált módszerről van szó, amely egy hagyományos PC-n maradéktalanul megállja a helyét. Az érintőképernyős eszközökön – ahol a felhasználó nem egy pixelnyi méretű ponttal (az egérkur-

zorral), hanem az ujjával navigál – viszont már nem annyira.

- Nincs igazi multiplatform implementáció: a legtöbb asztali blogkliens természetesen Windows-rendszereken érhető el, a .NET keretrendszerre implementálva. Ez csak akkor jelent multiplatform megoldást, ha csak Windows rendszerekről van szó, az ilyen programok más operációs rendszereken (Linux, Android, Mac OS X, iOS) sajnos nem mindig használhatók. Néhány blogkliens, különösen a nyílt forráskódú változatok a másik végletet képviselik, ugyanis kizárólag Linux vagy BSD rendszereken működőképesek.
- Stabilitási és teljesítménybeli problémák: a fenti szempontra megoldást tudnak nyújtani a Java-alapú alkalmazások, ezek azonban teljesítmény szempontjából a közelében sincsenek a natív alkalmazásoknak, egy esetben pedig még a .NET alapúaknak sem.

A programom tervezésénél igyekeztem ezeket a problémákat is szem előtt tartani, illetve legjobb tudásom szerint megoldani.

1.5. A programban megvalósítani kívánt célok

A program képességei a tervek szerint a következők lesznek:

- legalább egy publikációs protokoll támogatása
- cikkírás lehetőségének biztosítása
    - bejegyzések írása, módosítása, törlése
    - offline cikkírás lehetősége (a szinkronizálás akkor történhet meg, ha van internetkapcsolat)
    - WYSIWYG szerkesztőterület, és HTML kód szerkesztését biztosító szerkesztőterület
    - bejegyzések előnézete
    - kész szövegek beillesztése (*.txt szövegfájlokból)
- kész cikkek kezelése:
    - helyi biztonsági másolat készítése a cikkekről, egy adatbázisban, saját formátumban, kérésre visszaállítás
    - cikkek különböző feltételek szerinti szűrése, keresés a cikkek tartalmában

- egyéb tulajdonságok:
    - jegyzettömb
    - címkék és kategóriák kezelése

Véleményem szerint ezek azok az alaptulajdonságok, amelyeket egy asztali blogkliensnek teljesíteni kell. Mivel a program célja egy gyakran használt webes technológia szemléletes bemutatása, a fent felvázolt tulajdonságok elégnek tűnnek. Természetesen a program továbbfejleszthető, ennek lehetséges irányait később megemlítem.

2. A program megvalósításához szükséges eszközök

Ahhoz, hogy a programot megvalósíthatónak minősítsük, meg kell határoznunk azokat az eszközöket, amelyekkel a problémát jó eséllyel meg tudjuk oldani. A szakdolgozat témájának kiválasztásával egy időben kezdtem el keresni a megvalósítás módját. A következő megállapításokra jutottam:

- Több olyan hálózati protokoll is létezik, amelynek segítségével egy blog tartalmát külső, harmadik féltől származó programokkal lehet kezelni. A készülő programot ennek megfelelően legalább egy ilyen protokoll támogatására fel kell készíteni.
- A programozás egyik törvénye szerint nem szükséges mindent újra feltalálni, lehetséges más fejlesztők már kész munkáját is használni. A programban mindenképpen szükség lesz adatbáziskezelő, hálózati kommunikációt lehetővé tévő könyvtárakra, valamint egy GUI könyvtárra, a felhasználói felület felépítéséhez.

Ezeket (szerencsére) nem kell a nulláról megírnom, többféle kész megoldás közül is választhatok, különálló programozói könyvtárak, vagy egységes programozói keretrendszer formájában.

Miután ezeket meghatároztam, számba vettem az összes alternatívát, amivel találkoztam. Ezek után kiválasztottam nekem megfelelő publikációs protokollt, és egy keretrendszert. A következő fejezetekben szeretném a választásomat megindokolni.

2.1. Kommunikáció biztosítása a blogmotor és a program között – a publikációs protokollok

A modern tartalomkezelő rendszerek rendszerint biztosítanak valamilyen interfészt, amelynek segítségével más tartalomkezelőktől, vagy egyéb programoktól tudnak kéréseket fogadni, illetve ezekre a kérésekre válaszolni. Az ilyen kommunikációs formára kitűnő példa az XML-RPC, az RSS[8], vagy az Atom[9] szabvány. Az ilyen formátumokat ismerő programok képesek arra, hogy a tartalomkezelő bejegyzéseit lekérdezzék, és azt a felhasználó számára is „élvezhetővé” tegyék. Az ilyen és ehhez hasonló interfészek olvasása szinte mindig lehetséges, de mi van akkor, ha új tartalmat szeretnénk feltölteni a tartalomkezelőbe? A készülő programnak ugyanis elég gyakran kell majd ehhez folyamodni. A fent említett protokollok, rendes, összefoglaló nevükön, a publikációs protokollok erre, vagyis a fordított irányú kommunikációra is képesek.

8 http://tools.ietf.org/id/draft-nottingham-rss2-00.txt

9 RFC 5023: http://tools.ietf.org/html/rfc5023

A világ jelenleg több publikációs protokollt is ismer, a legnépszerűbbek a távoli eljárás-híváson alapuló publikációs protokoll, vagyis az XML-RPC[10], valamint az Atom Publishing Protocol (AtomPub).

A programomban az XML-RPC protokollt fogom felhasználni, egyrészt, mivel ez minden egyéb publikációs protokoll őse, másrészt, mivel az implementálása egyszerűbbnek tűnik, mint a többieké.

2.2. Az XML-RPC protokoll ismertetése

Az XML-RPC egy univerzális protokoll, amely internetes szerverek által biztosított eljárások HTTP protokollon keresztüli meghívására szolgál. Blogprotokollról akkor beszélhetünk, ha ezek az eljárások egy blogmotor külső programozói interfészéül szolgálnak. A továbbiakban XML-RPC protokollként fogok erre az esetre hivatkozni, az egyszerűség kedvéért, valamint a szakirodalommal összhangban.

Miért pont ezt a protokollt kívántam megvalósítani a programban? Választásom okai a következők:

- az XML-RPC protokollt olyan híres-neves blogmotorok támogatják, mint például a Wordpress, a Serendipity, a Drupal, vagy a Movable Type, ennek következtében kellőképpen elterjedt, mondhatni, de facto szabvány,
- más elterjedt blogprotokollokhoz (Atom Publishing Protocol, Google Blogger 2.0) képest sokkal egyszerűbben implementálható,
- az XML-RPC protokoll által lefektetett alapelveket más protokollokban is felhasználták, így ez a protokoll a többi protokoll kvázi „ősének” tekinthető. Ez kedvező a program továbbfejlesztése szempontjából.

Az XML-RPC protokoll az egyik első, széles körben használt blogprotokoll. Mai formáját kicsivel több, mint egy évtizedes fejlődés során érte el. Az újabb és újabb generációs blogmotorok mindig egy kis pluszt tettek hozzá, így mindig teljesíteni tudta az egyre nagyobb igényeket.

2.3. Az XML-RPC protokoll története

10 http://www.xmlrpc.com/

A protokollt 1998-ban Dave Winer dolgozta ki a UserLand Software és a Microsoft alkalmazásában. Szabadalmi bejegyzése jóval később, 2006 áprilisában történt, a szabadalom jelenleg a Virginia állambeli WebMethods vállalat tulajdonában van. Az XML-RPC továbbfejlesztéséből született meg az igen elterjedt SOAP protokoll [8].

ábra 2. Dave Winer

A protokoll lényege, hogy a kliens a szerver által biztosított távoli eljárásokat (remote procedures) a http protokollon keresztül, XML formátumban előállított kérésekkel hívja meg, az eredményt tartalmazó válasz szintén XML formátumban érkezik.

Egy XML-RPC kérés a függvény nevéből és paramétereiből áll. A paraméterek erősen típusosak. Példaként álljon itt egy XML-RPC hívás kódja (forrás: Wikipedia [8]):

```
<?xml version="1.0"?>
<methodCall>
  <methodName>examples.getStateName</methodName>
  <params>
    <param>
        <value><i4>40</i4></value>
    </param>
  </params>
</methodCall>
```

A kérést a <methodCall> tag fogja közre. A <methodName> tag tartalmazza a távoli eljárás pontos nevét. Ez semmiképpen sem lehet üres. A <params> tag a paraméterek listáját tartalmazza, ez lehet üres is, ha a függvény nem vár paramétert. Minden paraméter a <param> tag-ben található, értékét a <value> tag zárja közre. Kötelező megadni a paraméter típusát, ez a fenti példában egy 4 bájtos (32 bites) előjeles egész szám, mint ahogy azt az <i4> tag is jelzi.

A paraméterek típusai a következők lehetnek

- tömb (array)
- base64 kódolású bináris adat (base64)
- logikai (boolean)
- ISO 8601 szabványú dátum és idő (dateTime)
- egész szám (i4 vagy integer)
- sztring (string)

- struktúra (struct)
- null érték (nil)

2.4. Az XML-RPC, mint blogprotokoll

2.4.1. A Blogger API első változata

Felhasznált források*:* Isaac Yassar blogbejegyzése [5] a Blogger történetéről, valamint a magyar [4] és az angol [3] nyelvű Wikipedia vonatkozó szócikke.

Az XML-RPC blogprotokollként történő hasznosítását igen hamar, 1999-ben kezdték. Ekkor indult hódító útjára a Pyra Labs által fejlesztett Blogger. A Blogger volt az egyik első ingyenes blogszolgáltatás, magánszemélyek részére. (Az igazsághoz hozzátartozik, hogy nem volt egészen ingyenes, bizonyos szolgáltatásokat csak a prémium felhasználók vehettek igénybe, némi pénzösszegért cserébe) A Blogger szolgáltatás részeként lehetőség volt külső kliensprogrammal csatlakozni egy Blogger bloghoz, ez az XML-RPC protokoll segítségével valósult meg. (Az Atom Publishing Protocol ekkor még nem is létezett, az RSS tervezete is csak ekkor kezdett kialakulni.)

A Blogger által megvalósított XML-RPC hívások igen egyszerűek voltak, csak néhány alapvető funkciót támogattak, pl. bejegyzés beküldése vagy módosítása (törölni még nem lehetett), felhasználó account-jában rögzített adatok lekérdezése.

A Pyra Labs csapatát, és ezzel együtt a Blogger szolgáltatást 2004-ben felvásárolta a Google. Ezután a szolgáltatást az alapoktól kezdve újraírták, hogy a Google saját GData API-jába illeszkedjen. Így dobták az XML-RPC protokoll támogatását, a Blogger mai verzióiban ez a protokoll már nem támogatott. Ennek ellenére más blogmotorokban a mai napig használják, elsősorban történelmi okokból.

2.4.2. Az újjászületés: a MetaWeblog API

Felhasznált források: Dave Winer leírása [11] az xmlrpc.com weboldalon, illetve az angol Wikipedia vonatkozó szócikke [12].

Dave Winer, akinek az XML-RPC protokollt is köszönhettük, 2002-ben dolgozta ki a MetaWeblog API-t. Az eredeti cél a Blogger API korlátainak feloldása volt.

A MetaWeblog képes törölni is a bejegyzéseket, de a beküldéshez és a módosításhoz is újfajta módszert biztosított. A bejegyzéseket nem egyszerű szövegként, hanem struktúraként definiálta, az összetettebb adatszerkezet nagyobb tudást eredményezett. A bejegyzéseknek

már címet is lehetett adni, illetve megjelentek a kategóriák. (ezek nagyjából ugyanazt a szerepet töltötték be, mint manapság a címkék)

A MetaWeblog eredetileg három új függvényt definiált, ezek a metaWeblog.newPost(), metaweblog.editPost() és a metaweblog.deletePost(). Később, amikor a Blogger dobta a saját protokollját, az eredeti Blogger API függvényei is a MetaWeblog részévé váltak.

2.4.3. Movable Type és Wordpress: az XML-RPC ma

A Movable Type[11] egy igen népszerű blogmotor, jelenleg a Six Apart nevű cég fejleszti. Első változata 2001. október 8-án jelent meg. 2007. december 12. óta nyílt forráskódú program, GPL liszensszel. Perl programnyelven készült.

Legjelentősebb újítása a trackback-ek kezelése, melyet azóta számtalan blogmotor átvett. A trackback-ek speciális hivatkozások, amelyekkel a blogmotor nyilván tudja tartani, melyek azok a blogok, amelyek rá hivatkoznak. Támogatja a statikus oldalak létrehozását, a kategóriák és a címkék kezelése is szét lett választva.

A Wordpress[12] a jelenlegi legnépszerűbb nyílt forráskódú, PHP alapú blogmotor, de képességei miatt egyre gyakrabban alkalmazzák általános tartalomkezelő rendszerként.

Mindkét tartalomkezelő rendszer aktívan használja az XML-RPC protokollt, illetve az általuk biztosított fejlesztői keretrendszerben külön metódusok szolgálnak az XML-RPC függvények meghívására.

2.6. A fejlesztéshez használt keretrendszer

A program céljainak kielégítő megvalósításához szükség van egy olyan programozói keretrendszerre, amely a tervezés elején meghatározott céloknak megfelel, ezek közül (számomra) a legfontosabb, hogy legyen minden jelentősebb platformon használható. Számtalan jobb-rosszabb megvalósítás létezik erre a problémára, az én választásom ezek közül az egyik legjobbnak tartott eszközre, a Qt keretrendszerre esett. A most következő fejezettel szeretném bemutatni, miért.

A fejezet írása során használt legfontosabb forrásom a „C++ GUI Programming with Qt 4” című könyv [17] második kiadása, melynek szerzői Jasmine Blanchette és Mark Summerfield, illetve az angol nyelvű Wikipedia vonatkozó szócikke [16].

11 http://www.movabletype.org/

12 http://wordpress.org/

2.7. A Qt ismertetése

A Qt egy multiplatform programok fejlesztésére használható, igen hatékony, modern, objektum-orientált keretrendszer. Eredetileg a C++ nyelvhez készült, de mára rengeteg más programnyelvi környezetben is használhatjuk. Hogy miért választják a programozók szívesen a Qt-t kisebb-nagyobb programjaik megírásához?

„A Qt azért sikeres, mert a programozók kedvelik” - írta Matthias Ettrich, a KDE projekt alapítója, az egyik legismertebb Qt fejlesztő a Blanchett-Summerfield könyv előszavában [17]. Ugyanis a Qt keretrendszer használatával tényleg egyszerűbbé válik a (C++) programozók élete.

3. ábra. A Qt hivatalos logója

Az elsődlegesen támogatott nyelvi platform mind a mai napig a C++ nyelv, amit több érdekes dologgal bővít ki:

- sajátos eseménykezelési metódus, a szignál-szlot mechanizmus segítségével, a C stílusú callback függvények helyettesítésére
- QVariant osztály, amely egyfajta dinamikus típus
- saját primitív adattípusok, melyek memóriabeli mérete minden támogatott platformon azonos
- közös ősosztály, melynek QObject a neve
- saját tároló (konténer) típusok, és a kezelésükre szolgáló algoritmusok, a C++ beépített STL könyvtára helyett
- új nyelvi szerkezetek: foreach ciklus, szignálok és szlotok

A Qt legfontosabb tulajdonságai:

- több operációs rendszer támogatása: a Qt a Windows/Mac OS X/Linux hármas mellett a lelkes közösségnek köszönhetően olyan platformokon is elérhető, mint például a BSD, a Haiku, az Amiga OS stb. Emellett olyan mobil operációs rendszereken is használható mint például az Android, a Symbian, a webOS vagy az Apple iOS.
- több programnyelv támogatása: a Qt hivatalosan a C++, illetve a Java nyelv segítségével programozható, de félhivatalos (közösségi) támogatással rendelkezik Free-

Pascal, Python, Ruby, PHP, C#, Perl nyelvekre

- támogatja a többnyelvű alkalmazások fejlesztését: a Qt keretrendszer belül mindenhol az UTF-16 karakterkódolást használja, ezt konvertálja arra a karakterkódolásra, amelyet a futtató operációs rendszer használ (Linux esetén például UTF-8-ra) Emellett a programban megjelenő sztringeket egy szótárállomány segítségével mindig az aktuális nyelvhez igazítja, ezt a szótárállományt egy külön program segítségével módosíthatjuk, így a program fejlesztője és fordítója egymástól külön is dolgozhat. Az éppen használt nyelv futási időben, a program újraindítása nélkül is megváltoztatható.
- A 4.7-es verziótól kezdve a keretrendszer részét képezi a QML nyelv, egy Javascript-szerű deklaratív programnyelv, látványos felhasználói felületek létrehozásához. A QML segítségével szétválaszthatjuk a felhasználói felületet és a program működési logikáját. Ez a technológia sokban hasonlít a Microsoft Silverlight-hoz, vagy az Adobe Flash-hez.

2.7.1. Története

A Qt története 1991-ben kezdődött [17]. Két fiatal svéd programfejlesztő, Haavard Nord és Eirik Chambe-Eng egy adatbázis alkalmazás kifejlesztését kapta feladatul. Az egyik legnagyobb probléma az volt, hogy három platformon (Windows 3.1, Mac OS, UNIX) kellett volna futnia a programnak, mindhárom platformon teljes értékű és azonos módon működő grafikus felhasználói felülettel. Ez akkoriban problémát jelentett, hiszen a három operációs rendszer háromféleképpen közelítette meg a grafikus interfész témakörét, három különböző programozói felülettel. Egy áthidaló megoldásra volt tehát szükség, amely eltünteti az operációs rendszerek különbözőségeit. Az ötlettel Haavard állt elő, aki megállapította, hogy egy objektum-orientált megjelenítőrendszerre van szükségük, amely mindhárom platformon ugyanazokat az építőköveket biztosítja.

4. ábra. Haavard Nord

5. ábra. Eirik Chambe-Eng

Az előkészületek után a Qt fejlesztése 1991-ben kezdődött az alaposztályokkal, amelyeket Haavard fejlesztett, a design-nal kapcsolatos munkálatokat Eirik vállalta. 1992-ben Eirik fejéből pattant ki a szignál-szlot mechanizmus alapötlete, amely mindmáig a Qt egyik különlegessége. 1993-ban a rendszer már képes volt grafikus objektumok megjelenítésére is.

A készülő grafikus rendszer 1994-ben kapta a Qt nevet. A névben szereplő „Q” a fejlesztők elmondása szerint nem jelent semmit, csak azért került a névbe, mert a Haavard által használt Emacs betűtípusában ez a betű különösen jól nézett ki [17]. A „t” pedig a „toolkit” szó rövidítése, egyúttal utalás az akkoriban gyakran használt „Xt” nevű grafikus eszközkészletre.

A fejlesztés egy ideig gyakorlatilag hobbiprojektként futott, egészen 1994 március 4-éig, amikor is bejegyzésre került Svédországban a Quasar Technologies nevű cég, amely a Qt további fejlesztését és terjesztését végezte. A cég később felvette a Troll Tech, majd a Trolltech nevet. A céget 2008. júniusában felvásárolta az ismert finn mobilkommunikációs vállalat, a Nokia, az akvizíciót 2008. januárjában jelentették be [18]. Miután a Nokia és a Microsoft szövetségre lépett, a Qt kereskedelmi licencelésének és támogatásának jogát a finn Digia kapta meg [19]. 2011. október 21-én jelentette be [20] Lars Knoll Qt fejlesztő a Qt Project nevű kezdeményezést, ezzel a Qt valódi szabad és nyílt forráskódú projektté vált, az ilyen projektekre jellemző fejlesztési és irányítási modellel.

A Qt 1.0-s verziójú kiadása 1996. szeptember 24-én jelent meg. A jelenlegi (ez természetesen a szakdolgozat írásának idejét jelenti) legfrissebb kiadás a 4.7.4-es verziószámot viseli, 2011. szeptember 1-i megjelenési dátummal, de már aktívan fejlesztik a sok szempontból forradalmi, 5.0-s verziót is.

2.7.2. A Qt manapság

A Qt mára keresztplatformos widgetkészletből komplett fejlesztői keretrendszerré fejlődött. Szinte minden felhasználási területre létezik hatékony megoldása. A nyílt forráskódú változat a grafikus felület osztályain kívül webes erőforrások menedzselésére, adatbáziskezelésre, vektorgrafikus képek kezelésére, XML alapú adatcserére is tartalmaz osztályokat. Ezenkívül saját hangrendszerrel, saját beépített, WebKit alapú böngészőmotorral, és OpenGL alapú háromdimenziós grafikai alrendszerrel is rendelkezik.

2.7.3. A Qt jelenlegi legfontosabb alkalmazási területei

A Qt-t számtalan vállalat használja különböző fejlesztéseihez. A Qt referenciái [21] között olyan vállalatok is szerepelnek, mint a Lufthansa Systems, a Samsung, a Panasonic, a Hewlett-Packard, vagy az Európai Űrügynökség (ESA). A világ minden részén felbukkanhat a Qt, beágyazott rendszerekben, katonai, légiközlekedési, orvosi rendszerekben, leggyakrabban természetesen valamilyen beágyazott rendszerekre tervezett Linux kernel „társaságában”.

Számtalan kereskedelmi szoftver létezik, amely bevallottan a Qt keretrendszerre épül. Ilyen például az Adobe Photoshop Elements és Photoshop Album, az Autodesk Maya, a Skype, a Google Earth, a Mathematica, vagy az Oracle Virtualbox [16].

A nyílt forráskódú szoftverek világában a Qt jelenlegi legjelentősebb felhasználási területét a KDE projekt[13] jelenti. A KDE projekt feladata (többek között), hogy a GNU/Linux és BSD alapú rendszerek számára egy teljes értékű, minden szempontból modern felhasználói felületet, alkalmazáscsomagot, illetve fejlesztői platformot biztosítson. A KDE ennek érdekében még több, saját fejlesztésű osztályt adott hozzá a Qt keretrendszerhez. A Qt és a KDE szimbiózisa példaértékű, a Qt fejlesztésében a KDE közösség is részt vesz, páran a Trolltech/Nokia alkalmazásában, mint például a KDE atyja, Matthias Ettrich. A KDE felhasználói felülete minden modern Linux-disztribúcióban megtalálható, a Kubuntu Linux, az OpenSUSE, és még jó pár Linux-disztribúció alapértelmezett felhasználói felülete. BSD rendszereken is népszerű, az igencsak figyelemre méltó PC-BSD rendszer is KDE-t használ, némi módosítással.

A szakdolgozat írásának idején jelent meg az Ubuntu Linux 11.10-es verziója. Ez az első kiadás, amelyben a 3D gyorsítóval nem rendelkező hardverekhez készült Unity 2D nevű felhasználói felület is helyett kapott. A Unity 2D-t is a Qt keretrendszerben, a QML nyelv felhasználásával készítették el [24].

A jövőben jelentős felhasználási terület lesz még a mobil eszközök területe. Bár korábban is voltak tervek a Qt mobil eszközökön való alkalmazására (lásd Qtopia), ezen a téren mégis a Nokia révén fejlődött rengeteget a Qt. A Nokia hányattatott sorsú, új generációs mobil operációs rendszere, a MeeGo is a Qt keretrendszert használja, a grafikus felület felépítéséhez és a fejlesztői platform biztosításához. A jelenleg még népszerűnek számító Symbian operációs rendszer is képes Qt-ben íródott programok futtatására, esetleges jövőbeni verziói pedig szintén a Qt-ra épülnek majd. De már bárki számára tesztelhetők azok a megoldások is, amelyekkel a Google Android, Apple iOS, illetve Mono/.NET platformokra is fejleszthetünk Qt-s alkalmazásokat.

2.8. A Qt használata

A történelmi összefoglaló után lássuk, mi fogott meg a Qt-ban, miért pont az általa biztosított lehetőségekkel kívántam élni.

Fontos megjegyezni, hogy terjedelmi, valamint praktikussági okokból csak a C++ nyelv

13 http://kde.org/

esetére térek ki, mivel a szakdolgozatom tárgyát képező program is ebben készült. A fejezetben felhasznált forrás a Qt hivatalos dokumentációja[22] és a Wikipedia szócikke [16].

2.8.1. Felépítése

A Qt úgynevezett modulokból épül fel. Mindegyik modulnak megvan a saját, jól körülhatárolt feladata. A modulok használata igen egyszerű, mivel sok szempontból hasonlítanak egymásra. A Qt nyílt forráskódú változata által szolgáltatott modulok a következőek:

QtCore – a Qt alap osztályait, típusait tartalmazza, minden Qt program működéséhez feltétlenül szükséges

QtGui – a grafikus felület osztályai, csak akkor szükséges, ha a programunkhoz grafikus felhasználói felületet is készítünk

QtNetwork – a hálózati kommunikáció teljes körű kezeléséhez szükséges osztályok

QtSql – relációs adatbázisok kezeléséhez szükséges osztályok

QtXml – XML adatállományok feldolgozásához szükséges osztályok

QtSvg – vektorgrafikus állományok kezeléséhez

QtOpenGL – 3D grafikai alkalmazásokhoz

QtScript – a felhasználói felületen alkalmazható szkriptnyelv, hasonló a Javascripthez

Phonon – a Qt saját hangrendszere

QtMultimedia – alacsony szintű multimédiás tartalmak kezeléséhez

Qt3Support – a Qt előző változatában írt programok könnyebb portolásához

QtDBus – egy ismert IPC mechanizmus Qt-s implementációja

Mint látható, a választék igen nagy. Természetesen ez csak egy kis részlet, az Interneten számtalan, harmadik fél által fejlesztett Qt modult találhatunk, ezek többnyire nyílt forráskódú megoldások.

Ahhoz, hogy egy modult a programunkban használhassunk, hozzá kell linkelnünk a programhoz. Ezt a projekt leíró állományában, a *.pro kiterjesztésű állományban tehetjük meg. (A *.pro állomány felépítését később részletezem)

2.8.2. Szignálok és szlotok

A Qt egyedi tulajdonsága a szignálok, és az ezek aktivizálódására reagáló szlotok alkalmazása. A modellt a C programnyelv hagyományos callback függvényeinek leváltására dol-

gozták ki, tulajdonképpen az eseménykezelés egy speciális, rugalmasabb módja.

A callback függvények használata nehézkes, mivel a programozónak függvényekre, eljárásokra hivatkozó mutatókkal kell bajlódni. Emellett a callback függvényeknek nem egyszerű a paraméterezése, az adatok átadása csak kerülőúton, globális változókkal vagy megint csak mutatókkal oldhatók meg.

Ezzel szemben a szignálok és a szlotok alkalmazása pofonegyszerű, ráadásul egyszerre több paramétert is átadhatunk a szlotnak. Mindez hatalmas könnyebbséget jelent a programozó számára.

A szignál/szlot modell elviekben a kivételkezelést is képes kiváltani, a két modell hasonlósága miatt. A megfelelő szignálokhoz való csatlakozás, és a szloton belüli hibakezelés sok esetben több, mint elég.

2.8.3. Szignálok

A szignál, mint ahogy a neve is mutatja, egy jelzés arról, hogy a programban valami történt. Ha valamilyen meghatározott esemény következik be a program futása közben (például a felhasználó egy nyomógombra kattintott, vagy megérkezett a válasz egy hálózati kérésre), az eseményhez tartozó szignál emittálódik. Ha ez történik, a program soron kívül meghívja azt a metódust (tehát tkp. a szignálhoz tartozó szlotot), amelyet a megfelelő metódussal, a QObject osztály connect() nevű metódusával a szignálhoz kapcsoltunk. Egy esemény hatására több szignál is emittálódhat, illetve egy szignálra több szlot is válaszolhat. Fontos megjegyezni, hogy a program futása (néhány kivételtől eltekintve) a szlot végrehajtása után pontosan ugyanarról a helyről folytatódik, ahol a szignál emittálásakor abbamaradt. Ez jelentős eltérés a kivételkezeléshez képest.

Természetesen nincs akadálya annak, hogy a saját magunk által írt osztályainkban is használjunk szignálokat. Ehhez azonban be kell tartanunk néhány fontos szabályt.

- az osztályunk mindenképpen a QObject, vagy ennek valamelyik leszármazott osztályának (pl. QWidget) leszármazottja kell, hogy legyen
- az osztály deklarációban el kell helyeznünk egy Q_OBJECT sort
- az osztálydeklarációban a szignálokat csak a megfelelő helyen, a signals kulcsszó után deklarálhatjuk

Saját szignáljaink emittálásához az emit kulcsszót használhatjuk, pl.

```
emit myAnswer(42);
```

Ilyenkor a szignálra reagáló szlot egy int típusú értéket kap, melynek 42 az értéke.

2.8.4. Szlotok

A szlot speciális metódus, egy (vagy több) szignál kezelését végzi. Szerepe hasonló, mint a kivételkezelés esetén a catch blokknak. Ha a hozzákapcsolt szignál emittálódik, a szlot automatikusan végrehajtódik, a szignál által átadott paramétereket is automatikusan megkapja, további feldolgozásra. Egy szlot bármilyen osztályban szerepelhet, és a „közönséges" metódusokhoz hasonlóan public, private, vagy protected hozzáférési jogkörrel is rendelkezhet. Ennek megfelelően a szlotokat az osztálydefinícióban a private slots, a protected slots, vagy a public slots kulcsszó után kell deklarálnunk.

Egy szignál és egy szlot összerendelése a connect függvény segítségével történhet, amely egyébként a QObject osztály statikus metódusa. Egy kapcsolat létrehozására példa:

```
connect(this, SIGNAL(myAnswer(int)), this, SLOT(tellTheResult(int)));
```

Ez a connect függvény klasszikus alakja. Az első paraméter egy pointer, arra az osztályra hivatkozik, amely a szignált emittálja. A második paraméter maga a szignál, egy speciális makró, a SIGNAL() paramétereként. A harmadik paraméter annak az osztálynak a mutatója, amelyben a szignálra reagáló szlot található. A negyedik paraméter pedig a szlot, a szignálhoz hasonlóan egy makró, a SLOT() paramétereként.

Programjainkban szlotokat is létrehozhatunk, de itt is van egy fontos játékszabály: a szlot (legalábbis C++-ban) nem lehet bármilyen metódus. Az osztálydeklarációban a szlotokat a megfelelő helyen, a slots: kulcsszó után kell deklarálni

Ha a programunkban szükséges, a szignálok és a szlotok közötti kapcsolatokat meg is szüntethetjük. Erre a disconnect() statikus metódus szolgál. Szintaxisa (az előbbi példánál maradva) a következő:

```
disconnect(this,SIGNAL(myAnswer(int)),this,SLOT(tellTheResult(int)));
```

A függvény paramétereinek jelentése megegyezik a connect() függvénynél tárgyaltakkal. A metódus ilyenkor csak a megadott szignál-szlot kapcsolatot szünteti meg.

A disconnect() másik alakja:

```
disconnect(this,SIGNAL(myAnswer(int)));
```

Ebben az esetben az összes szlot kapcsolat törlődik, amely a myAnswer(int) szignálhoz kötődött.

2.8.5. A Qt projekt leíró állománya, a *.pro fájl

Ahhoz, hogy egy Qt program lefordíthatóvá válhasson, egy leíró állomány szükséges,

amely definiálja a fordítás szabályait. A Unix-os világ hagyományai szerint erre a Makefile-ok használatosak. Egy Makefile szintaxisa viszont igen bonyolult, különösen egy nagyobb projekt esetén, amelynek kézzel történő módosítása jelentős terhet róhat a programozóra. E probléma áthidalására a Qt egy úgynevezett projekt leíró fájlt alkalmaz, melynek kiterjesztése *.pro.

Míg a Makefile erősen platformfüggő, a projekt leíró fájl szerkezetéből, és feldolgozásának menetéből fakadóan platformfüggetlen.

A projekt leíró állomány változó=érték párokból áll. Egy változónak több értéket is adhatunk, az egyes értékeket ilyenkor a \ karakterrel, majd egy új sor indításával választhatjuk el. Értékadásnál használható az = értékadó operátor, illetve a += hozzáfűző operátor is. A változók neve csupa nagy betű kell, hogy legyen, az értékek természetesen kis és nagybetűt egyaránt tartalmazhatnak.

Az érvényes változók a következők, a teljesség igénye nélkül:

- QT: a projektben használandó modulok felsorolása. A modulok nevei kisbetűsek, és nem szerepel bennük a Qt, tehát, ha a QtNetwork modult kívánjuk a programunkban használni, a változónak a network értéket kell adnunk
- TARGET: a fordítás során létrejövő bináris állomány neve. Unix rendszereken ez a program teljes neve lesz, Windows esetén természetesen ehhez még az *.exe kiterjesztés is hozzákerül
- TEMPLATE: megadja, hogy a létrejövő bináris állomány milyen típusú legyen. Ha futtatható állományt szeretnénk, a változó értéke app kell, hogy legyen, programkönyvtár esetén pedig lib.
- SOURCES: a projektben szereplő, *.cpp kiterjesztésű forrásállományok felsorolása
- HEADERS: a projektben szereplő *.h kiterjesztésű header állományok felsorolása
- FORMS: ha a programunk grafikus felületet is használ, és a felületet *.ui fájlokban adtuk meg, ezeket az állományokat itt kell felsorolni
- RESOURCES: a programban használt erőforrás-állományok felsorolása
- LIBS: a programunk által használt külső programkönyvtárak felsorolása
- TRANSLATIONS: ha a programunkat több nyelv támogatására is felkészítettük, a Qt Linguist-tal létrehozott fordítási állományokat itt kell megadnunk

Ezek alapesetben mind platformfüggetlen opciók. Ha szeretnénk platformspecifikus szabályokat is megadni, azt is megtehetjük,

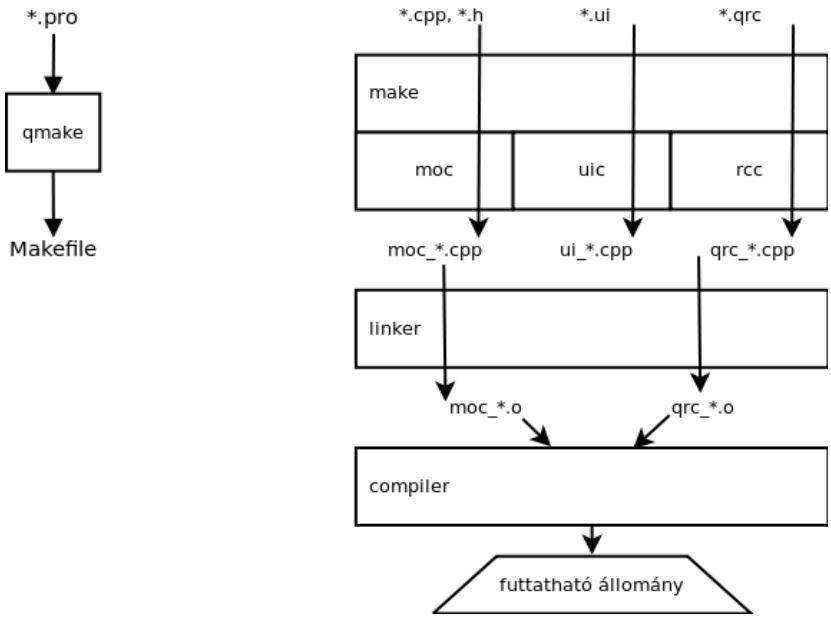
```
platform {
   szabályok
}
```

alakban. A platform lehetséges értékei: unix, win32.

A projekt leíróban megjegyzések is elhelyezhetők, ezeket a # karakterrel kell kezdeni. A # karakter utáni részt a qmake figyelmen kívül hagyja.

2.8.6. Egy Qt program fordítása

Ahhoz, hogy a Qt által bevezetett különleges szintaktikai elemeket használni tudjuk, a fordító oldaláról némi trükközés szükséges. Ezeket a nyelvi elemeket ugyanis (természetesen) a C++ fordítók semelyike sem támogatja. A Qt fejlesztői megtehették volna, hogy egy saját C++ compilert írnak, ez azonban több fejlesztői erőforrást vont volna el az érdemi munkától, mint amennyi előnyt jelentene. A Qt esetében ezért egy eléggé furfangos megoldást választottak a programozók.



ábra 6. Qt projekt fordítása C++ nyelv esetén

Egy C++ nyelven írt Qt program fordítását mutatja be a 8. ábra. Minden Qt programhoz tartozik egy *.pro kiterjesztésű állomány, a már korábban ismertetett projekt leíró állomány. Ebből a fájlból a qmake program egy szabványos C++ típusú makefile-t készít, amit a make program már képes értelmezni. A qmake által generált makefile először a moc, az uic, illetve a qrc programokat hívja meg, a program tényleges fordítása csak ezután történhet meg.

A moc jelentése: Meta Object Compiler. Feladata, hogy az eredeti forrásállományban található Qt-specifikus kulcsszavakat (mint amilyen például az emit, a foreach ciklus, a SIGNAL(), vagy a SLOT()) C++ nyelvű, azonos funkcionalitású programrészekkel helyettesítse. Az így létrejövő állomány neve elé a moc_ szócska kerül (például, ha egy osztályunk definíciója az frmMainWindow.cpp állományban található, a moc futása után a fordítható állomány neve moc_frmMainWindow.cpp lesz).

Az uic program a Qt Designer-ben létrehozott formokat hozza fordítható állapotra. Ezen állományok kiterjesztése *.ui, XML formátumban tartalmazzák a form leírását. Az uic feladata, hogy ezekből az XML leírásokból C++ nyelvű osztálydefiníciót készítsen. A létrejött állományok neve elé egy ui_ prefix kerül. Tehát, ha az előbbi formunkhoz egy frmMainWindow.ui fájl is tartozik, a létrejövő állomány neve ui_frmMainWindow.h lesz.

Az rcc program a Qt erőforrás-állományainak fordítását segíti. Egy erőforrás-állomány leginkább különböző formátumú képfájlokat, szöveges állományokat tartalmazhat, XML formátumban. Kiterjesztése *.qrc. Az rcc az állományból egy object fájlt generál, amely fordítás során összefűzhető a futtatandó állománnyal.

Miután sor került a forrásállományok előfordítására, a folyamat úgy folytatódhat tovább, mint bármilyen más C++ program fordítása esetén.

2.9. A Qt legfontosabb osztályai

2.9.1. QObject

A Qt keretrendszer a C++ nyelvet a közös ősosztály fogalmával is kiegészíti. A legtöbb Qt osztály egyetlen közös őstől, a QObject-től származik. A QObject biztosítja a lehetőséget például a szignálok és a szlotok használatára. Ha Qt programunkban egy olyan osztályt kívánunk létrehozni, amelyben szignálokra és szlotokra van szükség, az osztályt feltétlenül a QObject osztályból (vagy természetesen annak valamelyik leszármazottjából) kell származtatnunk.

Ezt mutatja be a következő egyszerű példa: legyen egy osztályunk, amely két egész számot ad össze, az eredményt pedig egy szignál segítségével küldjük el további feldolgozásra. A példa gyakorlati haszna igen minimális, de arra tökéletes, hogy egy Qt osztály deklarációját bemutassam.

```
class PrimitiveCalculator : public QObject {
Q_OBJECT
private:
        int num0;       // az összeadás egyik tagja
```

```
        int num1;       // az összeadás másik tagja
public:
        PrimitiveCalculator(int num0_0, int num1_0, QObject *parent=0);
        void Addition();
signals:
        void result(int res);
public slots:
        void processResult(int res);
};
```

Nézzük szép sorjában. A PrimitiveCalculator nevű osztály public típusú örökléssel származtatható a QObject osztályból, ebben semmi szokatlan nincs. Annál inkább a deklaráció második sorában: a Q_OBJECT egy makró, amely a Qt előfordítójának, a qmake-nek szól. Arra utasítja, hogy a megfelelő kiegészítő C++ kódot illessze be az osztálydefinícióba, így lehetővé téve az osztály megfelelő működését. (Ennek menetét lásd a qmake ismertetésénél). Az adattagok (az összeadandók), a konstruktor és az érdemi munkát végző Addition() függvény után következnek a szignálok és a szlotok deklarációi. A result szignál egy paraméterrel rendelkezik, amely az összeadás eredményét tartalmazza. Az osztály egy szlotot is tartalmaz, amely public hozzáférésű, és processResult() névre hallgat. A feladata az eredmény feldolgozása, vagy kiíratása a képernyőre. Itt is megjegyezném, hogy egy szignálnak és a hozzá csatlakozó szlotnak nem kötelező ugyanazon osztály tagjának lenni.

2.9.2. QWidget

A Qt második legfontosabb osztálya. Minden widget közös őse, vagyis minden osztály, amely valamit rajzol a képernyőre, a QWidget leszármazottja. Olyan adattagokat és metódusokat tartalmaz, amelyek minden widget működéséhez elengedhetetlenül szükségesek. Adattagjai a widget méretére, elhelyezkedésére és egyéb alapvető tulajdonságaira (engedélyezettség, láthatóság, tooltip szöveg, „what's this” súgószöveg stb.), metódusai ezek módosítására-lekérdezésére, illetve a widget kirajzolására, megjelenítésére vonatkoznak. Rengeteg virtuális metódust is tartalmaz, amelyek átdefiniálásával többnyire a felhasználói eseményekre (billentyűleütés, egérrel kapcsolatos események) reagálhatunk. Saját widgeteket is létrehozhatunk, a QWidget leszármaztatásával, és a megfelelő virtuális tagfüggvények definiálásával.

2.9.3. QApplication

A QApplication osztály feladata a grafikus felülettel rendelkező program vezérlése, a legfontosabb tulajdonságokhoz való hozzáférés biztosítása. A program indításakor inicializálja a grafikus felülelet, majd elindítja a főhurkot. Kezeli a program beállításait, a vágólapot. Képes a munkamenetek kezelésére is, ez biztosítja, hogy a program kilépéskori állapotát a

számítógép újraindítása vagy valamilyen rendszerhiba után a desktop rendszer visszatöltse (persze, amennyiben képes erre). A rendszeren egyszerre csak egy példány létezhet belőle, még akkor is, ha a hozzá tartozó program több példányban fut.

2.9.4. QVariant

Egy kvázi dinamikus típus. Bármilyen típusú adatot képes tárolni, az adat a kívánt formátumra konverziós metódusokkal alakítható. Elsősorban adatbázis-műveleteknél használható, a lekérdezések értékeinek megfelelő formátumra alakítására, de használható konverziós műveleteknél is, a C++ saját *to*() függvényei helyett.

Például egy 32 bites egész számot a következőképpen alakíthatunk QString-gé, amely a Qt saját, speciális sztringtípusa:

```
...
QVariant szam(42);
QString valaszom=szam.toString();
...
```

2.10. A programhoz szükséges osztályok

A program előreláthatóan a következő dolgokat kell, hogy véghezvigye:

- hálózati kommunikáció: a távoli blogmotorral való kommunikáció az Interneten keresztül, HTTP protokollt használva történik, ezért szükség lesz olyan osztályokra vagy metódusokra, amelyek képesek ezt a protokollt kezelni
- XML feldolgozás: a blogmotor és a program közötti kommunikáció XML üzenetek küldésével fog lezajlani, ezért szükség lesz olyan metódusokra is, amelyekkel gyorsan és hatékonyan lehet ezeket az üzeneteket feldolgozni, belőlük az adatokat kinyerni
- adatbázis-kezelés: a távoli blog adatait, bejegyzéseit helyileg, a felhasználó számítógépén is tárolni kell, ennek legegyszerűbb módja egy pillekönnyű adatbázis, mint amilyen az SQLite. Ehhez persze olyan osztályok is kellenek, amelyekkel ilyen típusú adatbázisokat lehet kezelni.

Miután tanulmányoztam a Qt keretrendszer képességeit, megállapítottam, hogy a rendelkezésre álló osztályok a fenti kritériumokat teljesítik, a Qt alaptelepítésével kompromisszummentesen meg tudom oldani a problémát. Nem szükséges külső, kiegészítő könyvtárak alkalmazása.

A következőkben azokat az osztályokat szeretném röviden bemutatni, amelyeket a prog-

ramban használni tudok. Természetesen nem fogom felsorolni az összes osztályt (ez eléggé sokáig tartana), csak a legfontosabbakat említeném. Ehhez most is a Qt keretrendszer dokumentációját [17] [22] fogom felhasználni.

2.10.1. Hálózatkezelő osztályok

QNetworkAccessManager

A Qt hálózatkezelő osztálya, a QtNetwork modul része. A HTTP protokollon keresztüli üzenettovábbításra alkalmas, de csak GET és POST kérés küldésére képes. Kezeli a proxyn keresztüli adattovábbítást, valamint a titkosított kapcsolatokat.

Működése az AJAX-éra hasonlít: egy kérés küldése külön szálon zajlik, a válasz megérkezését egy void finished(QNetworkReply*) nevű szignál jelzi, melynek paramétere tartalmazza magát a választ.

Közvetlenül csak aszinkron kéréseket lehet küldeni, de némi trükközéssel lehetséges szinkron kérések előállítása is.

QNetworkRequest

A kérést reprezentáló osztály. Használatához mindenképpen be kell állítanunk azt az URL címet, ahová a kérést küldeni szeretnénk, ezt a void setUrl(QUrl&) metódussal tehetjük meg. Lehetséges még a kéréshez saját fejléc adatokat (HTTP headers) is csatolni, ezt pedig a setHeader(...), vagy a void setRawHeader(...) metódusok valamelyikével tehetjük meg. A kérés titkosítása is megoldható; ha az SSL titkosítási metódust választanánk, a Qt ezt is támogatja, a QSslConfiguration osztályon keresztül. Az éppen használt SSL beállításokat a setSslConfiguration(QSslConfiguration *) metódussal állíthatjuk be.

QNetworkReply

A választ reprezentáló osztály. Tartalmazza a HTTP kérésre érkezett választ, vagy ha a kérés nem volt sikeres, a hibaüzenetet. A választ egy QString típusú sztringbe, vagy I/O bufferbe is továbbíthatjuk.

A dokumentáció szerint nem közvetlenül nem példányosítható, csak a QNetworkManager hozhatja létre. A továbbítás közben történt esetleges hibát a NetworkError error() függvénnyel kérdezhetjük le. A válasz tartalmát a read(), readLine(), vagy readAll() függvények valamelyikével olvashatjuk be, ezek a függvények sorrendben egy bájtot vagy egy karaktert, egy sort, vagy a teljes választ adják vissza.

2.10.2. XML adatok feldolgozása

A Qt keretrendszer támogatja mind a DOM, mind a SAX algoritmusokat. Emellett egy saját algoritmust is ismer, amely a SAX egyszerűsített változata, ennek neve QXmlStreamReader.

Mivel én a DOM modellt ismerem a legjobban, ezért a programomban is ezt fogom alkalmazni. A DOM fát a Qt-ben a következő osztályokkal lehet modellezni.

QDomDocument

Egy teljes XML dokumentum reprezentációja. Használatához először meg kell adni az adatforrást, ahonnan az XML dokumentum érkezik. Ez lehet egy QString, egy input/output puffer, vagy akár egy másik QDomDocument típusú objektum is. Ezt a setContent(...) nevű, többszörösen átdefiniált függvénnyel lehet megtenni.

A QDomDocument osztályból példányosított objektum leggyakrabban használt metódusa a QDomElement documentElement(), amely az XML dokumentum gyökér-elemét adja vissza.

QDomNode

Az XML dokumentum egy csomópontját tárolja. A csomópont adataival (vagyis a tag nevével, attribútumaival, tartalmával) nem foglalkozik, de a típusa könnyen megállapítható az isAttr(), isDocument(), isElement(), isText() stb. logikai típusú visszatérési értékkel rendelkező függvényekkel. A QDomNode csomópontot bármilyen típusú XML elemmé konvertálhatjuk, a megfelelő toAttr(), toDocument(), toElement(), toText() metódusokkal.

Mivel egyszerűbb adattípus, mint a hasonló QDomElement (lásd rögtön), a leggyakrabban a DOM fa bejárásánál használhatjuk. Az adott csomóponttal azonos szinten lévő, közvetlenül utána következő csomópontot a QDomNode nextSibling() metódussal határozhatjuk meg. A csomópont első gyermek csomópontját a QDomNode firstChild() metódus adja vissza.

QDomElement

Hasonló a QDomNode-hoz, de ez az osztály egy elemet reprezentál. Mivel ismert az elem típusa, így könnyen lekérdezhetjük a tulajdonságait. A tag nevét a QString tagName() függvénnyel, az attribútumainak értékét a QString attribute(const QString &name) függvénnyel, tartalmát pedig a QString text() függvénnyel tudhatjuk meg. Ezeknek természetesen beállító párjuk is létezik, ezek a void setTagName(QString), void setAttribute(QString

&name, <<érték>>), illetve a void setText(QString &).

A QDomElement is használható a DOM fa bejárására, de több memóriára van szüksége. Emiatt az XML feldolgozásnál tanácsos a QDomNode használata, a QDomElement-hez csak akkor forduljunk, ha a szükség megköveteli. Ilyen speciális eset az, ha egy csomópont azon gyermek csomópontjainak listáját szeretnénk megtudni, amelyek egy bizonyos tagnévvel rendelkeznek. Ezt egyébként a QDomNodeList elementsByTagName(QString &tagname) függvénnyel tehetjük meg. A QDomNodeList osztály egyébként egy QDomNode objektumokból álló lista, vagyis teljesen ugyanaz, mintha azt írtuk volna, hogy QList<QDomNode>.

2.10.3. Adatbázis-kezelés

A Qt a relációs adatbázisok elérésére lett felkészítve. A QtSql modul egy többé-kevésbé adatbázis-független elérést biztosít. Ahhoz, hogy a keretrendszer egy adatbázissal szót tudjon érteni, egy adatbázis driverre van szükség. A Qt szabad változata a legnépszerűbb szabad adatbáziskezelőkhöz tartalmaz drivereket; vagyis a MySQL/MariaDB-hez, az SQLitehoz, a PostgreSQL-hez, a többi RDBM-hez használhatjuk az ODBC drivert. Ha nagyon szükséges lenne, saját drivert is írhatunk, a Qt-nek ehhez is megvannak az eszközei.

QSqlDatabase

A QSqlDatabase osztály egy adatbázis elérését biztosítja. Használatakor először be kell töltenünk a megfelelő drivert, majd be kell állítanunk a DBMS eléréséhez szükséges adatokat. Például MySQL adatbázis esetén az adatbázis-szerver nevét, a port számát, az adatbázis (séma) nevét, valamint a bejelentkezéshez használt felhasználónevet és jelszót. SQLite adatbázis esetén csak az adatbázis nevét kell megadni, ez egyébként az adatbázis fájlrendszeren belüli elérési útvonala. A driver betöltését a QSqlDatabase példányosításánál kell megtennünk, az addDatabase(QString drivername, QString conn_name) statikus metódussal. A kapcsolathoz szükséges adatok beállításához a void setHostName(QString &), void setPort(int), void setDatabaseName(QString &), void setUserName(QString &), void setPassword(QString &) eljárásokat kell meghívnunk.

Ezt követően jöhetnek a lekérdezések, amelyeket a használt adatbázismotornak megfelelően kell megírnunk. A lekérdezéseket a QSqlQuery exec(QString &query) függvénnyel hajtathatjuk végre.

QSqlQuery

Az osztály egy SQL lekérdezés eredménytábláját, vagy nem végrehajtható lekérdezés esetén, a keletkezett hibát tartalmazza. A Qt keretrendszer, hasonlóan más adatbázis-kezelő megoldásokhoz, az eredménytábla feldolgozásához a kurzor mechanizmust használja.

A kurzor irányításához a következő metódusokat használhatjuk:

- bool first(): a kurzort a tábla első sorába állítja. Ha ez valamiért nem sikerült, visszatérési értéke false.
- bool last(): a kurzort a tábla utolsó sorába állítja.
- bool next(): a kurzort a tábla következő sorába állítja. Ha nincs több sor, visszatérési értéke false.
- bool previous(): a kurzort a tábla előző sorába teszi. Ha már nincs több sor, visszatérési értéke false.
- bool seek(int index, bool relative): a kurzort egy kiválasztott sorba pozicionálja. A hivatkozás lehet abszolút, vagy az éppen aktív sorhoz képest relatív. Ezt a relative logikai változó értéke szabja meg.

Az értékeket a QVariant value(int index) függvénnyel olvashatjuk ki. Csak szám szerinti hivatkozás van (tehát a táblázat oszlopának fejlécét nem használhatjuk), a számozás 0-val kezdődik. Az érték egy QVariant típusú változóban van, ezt a QVariant konverziós függvényeivel konvertálhatjuk a kívánt adattípusra.

A hibaüzenetet a QSqlError lastError() függvénnyel deríthetjük ki.

QSqlError

A lekérdezés végrehajtása közben történt hibát tárolja, hibakóddal, illetve a hiba teljes leírásával. A hibákat maga az adatbázis-motor, illetve az adatbázis-kezelő driver adja vissza, a Qt itt csak egyfajta közvetítőként játszik szerepet. Az egyes hibakódok jelentését a DBMS dokumentációjában lehet megtalálni.

Három fontosabb függvénye:

- int number(): a hiba kódja
- QString driverText(): a driver által küldött hibaüzenet szövege
- QString databaseText(): a DBMS által küldött hibaüzenet szövege

3. A program arculata

A készülő programnak, embrionális kor ide vagy oda, már szüksége van különböző ismertetőjelekre, amelyeknek köszönhetően a jövőben kitűnhet a „tömegből”. Ilyen ismertetőjel lehet például egy hangzatos név, vagy legalább egy kódnév, illetve egy embléma. Emellett meg kell határozni azokat az íratlan szabályokat is, amelyek alapján a program „kommunikálni” fog a felhasználóval. A következőkben a fenti dolgokat szeretnék nagyon röviden kitérni.

A programomat mandarina névre kívánom keresztelni. Ez természetesen egy kódnév, sem a népszerű déligyümölcs, sem annak spanyol elnevezése nem utal a program valódi funkciójára. Ellenben könnyen megjegyezhető, egyszerű rá brand-et építeni.

A program ikonja-emblémája a névnek megfelelően egy stilizált mandarin, amely mellesleg a képregényekből ismerős szövegbuborékokra próbál emlékeztetni, ezzel utalva arra, hogy a program elsődleges funkciója tulajdonképpen egy bizonyos fajta írásbeli kommunikáció megkönnyítése. Mivel nem vagyok professzionális grafikus (talán még amatőr sem), nehéz volt az alap formát megtalálnom, az ikon több koncepcióváltáson is átesett, mire elnyerte végleges formáját. Biztos vagyok benne, hogy egy profibb grafikus csodákat tudna belőle kihozni.

7. ábra. A program logója

A programban megjelenő szövegeket igyekeztem úgy megírni, hogy már abból is érződjön a program felhasználási területe. Mivel nem egy műszaki programról, hanem egy olyan felhasználói alkalmazásról van szó, amelynek elsődleges célközönsége a fiatalabb, interneten publikáló korosztály, szerettem volna elkerülni a magázódó, és ezzel együtt kicsit udvariaskodó fordulatokat. Helyette a tegezést választottam, amellyel egy sokkal közvetlenebb, sokkal barátságosabb hangnemet próbáltam megütni.

A szokványostól eltérő felhasználói felület kialakításáról a következő fejezetben részletesen fogok mesélni.

4. A felhasználói felület tervezése

Ebben a fejezetben szeretném bemutatni a mandarina program felhasználói felületét. A tervezéshez az Eclipse[14] technológiáin alapuló Wireframe Sketcher[15] programot használtam.

4.1. A felület alap koncepciója

A felhasználói felület megtervezésekor a következő szempontok szerint igyekeztem eljárni:

- a program interfésze szakítson a több évtizedes menüsáv-eszköztár-munkaterület-állapotsor sablonnal
- a program kezelése legyen a lehető legegyszerűbb
- a leggyakoribb funkciók bármikor egy-két kattintással elérhetőek legyenek
- csak azok a funkciók legyenek a felhasználó szeme előtt, amelyeket az adott szituációban ténylegesen használni tud és akar, egyetlen widget se foglaljon el feleslegesen helyet

Véleményem szerint olyan interfészt sikerült alkotnom, amely minden tekintetben megfelel a modern kori követelményeknek. Annak ellenére, hogy nem sokszor dolgoztam érintőképernyős eszközökkel, mégis megpróbáltam elképzelni, hogy nekem miként lenne kényelmes egy ilyen eszközre optimalizált programmal tevékenykedni.

A tervezésnél a következő programok felülete inspirált:

- Google Chrome[16]: ebben az ismert webböngészőben hasonló egygombos főmenü található, mint ami a mandarina főablakában
- Qt Creator[17], illetve Clementine Audio Player[18]: ebből a két programból (az előbbi a Qt keretrendszer saját integrált fejlesztői környezete, utóbbi pedig egy multiplatform zenelejátszó) érkezett az ötlet, hogy a felületet ne elsősorban ablakokra, hanem lapokra bontsam
- Wordpress[19]: a népszerű webes tartalomkezelő rendszer bejegyzésszerkesztőjének külalakját vettem alapul a mandarina bejegyzésszerkesztőjének megtervezésekor

14 http://www.eclipse.com

15 http://wireframesketcher.com/

16 http://www.google.com/chrome/

17 http://qt.nokia.com/products/developer-tools/

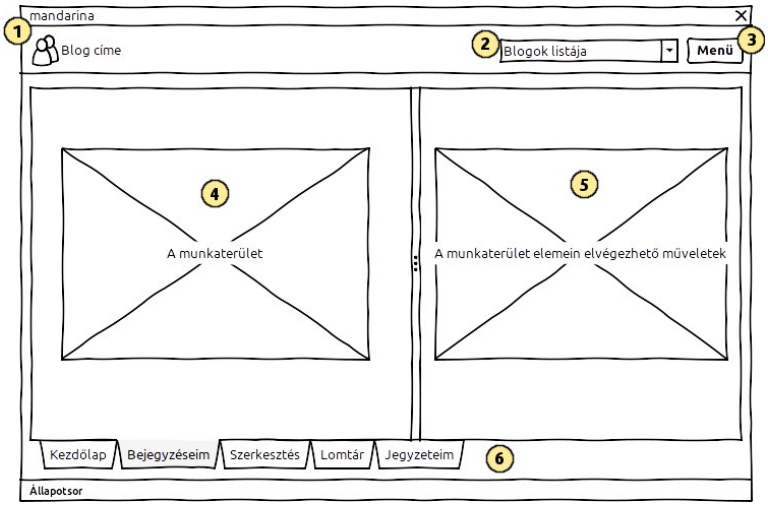
18 http://www.clementine-player.org/

19 http://wordpress.org/

4.2. A felület elemei

4.2.1. A főablak felépítése

A 10. ábrán az interfész sablonja látható, az említésre érdemes részeket számozással jelöltem.



ábra 8. A program interfészének sablonja

Az 1-es számmal jelölt widget a blog fejléce. Itt a blog ikonja illetve a felhasználó által megadott becenév jelenik meg. A blogikon egyúttal állapotjelzőként is funkcionál, függőben lévő, vagy már befejezett művelet sikerességéről is tájékoztat.

A 2-es jelű legördülő lista a blog lista. Itt érhetőek el azok a blogok, amelyeket a felhasználó a program „gondjaira bízott". A lista egy elemének tartalma a blog beceneve és a könnyebb azonosíthatóság érdekében, a blog ikonja.

A 3-as jelű nyomógomb a program főmenüje. Ez egy legördülő menüvel kombinált nyomógomb, ha nyomva tartjuk, megjelenik a főmenü, ha rákattintunk, akkor betöltődik a „Beállítások" párbeszédpanel.

A 4-es panel a program munkaterülete. Erre a területre kerülnek azok a dolgok, amelyekkel egy adott pillanatban foglalkoznia kell a felhasználónak. Például itt helyezkedik el a bejegyzések listája, vagy az éppen szerkesztett bejegyzés.

Az 5-ös panel a műveletsáv, amely a munkaterülethez tartozó műveleteket tartalmazza. A panel tartalma mindig változik, az adott szituációtól függően. A fenti példáknál maradva, ha a bejegyzéslistával dolgozik a felhasználó, a panelen a bejegyzések létrehozásához, szerkesztéséhez, beküldéséhez, vagy törléséhez kapcsolódó lehetőségek jelennek meg. Hasonlóképpen egy bejegyzés szerkesztésénél ezen a panelen jelenik meg a formázó „eszköztár".

A 6-os számmal jelölt terület az oldalsáv. A fülek elhelyezkedése és elrendezése a tervek szerint szabadon változtatható lenne.

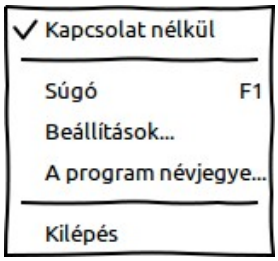
4.2.2. A főmenü

A 11. ábrán a tervezett főmenü látható.

A legfelső menüponttal lehet kikényszeríteni a kapcsolat nélküli munkamenetet. Elképzelhető ugyanis (desktopon szerencsére már ritkán, mobilinternettel viszont még előfordulhat ilyesmi), hogy a felhasználó kordában szeretné tartani az általa küldött vagy fogadott internetes adatmennyiséget. Ezen opció ki/be kapcsolásával ezt könnyedén megteheti.

A következő menüpontra kattintva elindul a program súgója.

A „Beállítások” menüponttal az azonos nevű párbeszédablakot indíthatjuk el. Ezen a párbeszédpanelen állíthatja be a felhasználó a program megjelenését (például az oldalsáv elhelyezkedését és méretét), az internetkapcsolathoz szükséges paramétereket (proxy szerver adatai), továbbá itt kezelheti a már regisztrált blogjait is.



ábra 9. A főmenü

A következő menüpontra kattintással hasznos információkat kaphatunk a programról és annak készítőjéről.

A legutolsó menüponttal kiléphetünk a programból.

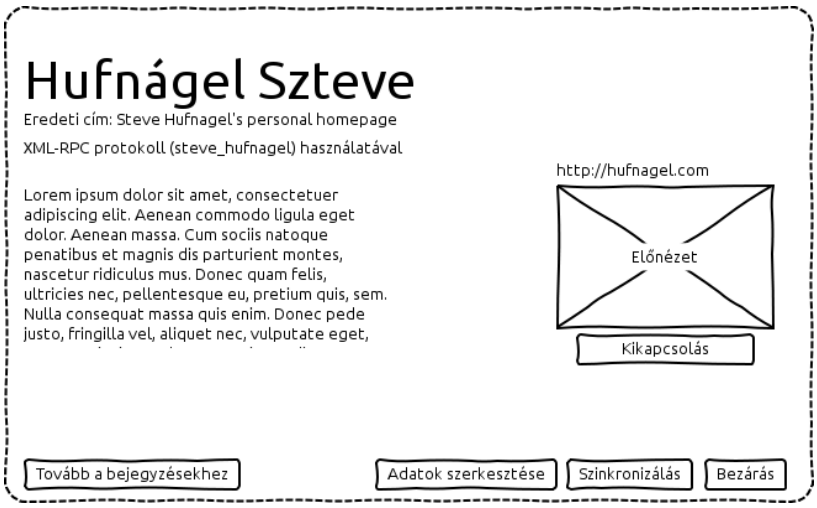
A felület vázlatos felépítésének ismertetése után most vegyük szemügyre az egyes konkrét használati eseteket.

4.2.3. A kezdőképernyő

Az oldalsáv legelső tagja, a program indításakor ez jelenik meg, a többi lap le van tiltva, egészen addig, amíg a bloglistából a felhasználó egy blogot ki nem választ.

A program indításakor a kezdőképernyő csak kevés elemet tartalmaz, a narancsszínű logón és egy rövid üdvözlő szövegen kívül még két gombot, az egyikkel egy új blog regisztrálható, míg a másikkal egy rövid, gyakorlatilag gyorstalpaló-jellegű súgó indítható.

Egy blog kiválasztásakor a kezdőképernyő megváltozik, a 12. ábrán látható külalakot veszi fel.



ábra 10. A főképernyő

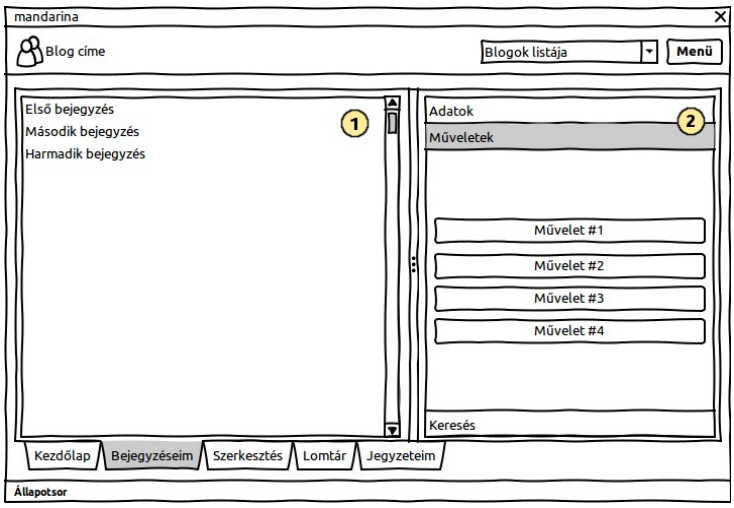
A képernyő bal oldalán a blog adatai láthatóak: a blog beceneve, az eredeti cím, a használt protokoll, a blogmotor azonosítója, valamint a blog leírása.

A jobb oldalon a blog URL címe alatt a blog miniatűr előnézete látható, amelynek megjelenítését a „Kikapcsolás” címkéjű, két állapotú gombbal (kapcsológomb) lehet ki- vagy bekapcsolni.

Az alsó részen négy nyomógomb látható. Az első „Tovább a bejegyzésekhez” feliratú gombbal a bejegyzéslistára lehet ugrani, igazából kényelmi funkciókat szolgál. Az „Adatok szerkesztése” gombbal a blog korábban megadott adatait tartalmazó párbeszédpanel indítható. A „Szinkronizálás” gombra való kattintás hatására elindul a távoli blog és a program blog adatbázisának szinkronizálása. A „Bezárás” gombbal az aktuális blog bezárható, a program alapállapotba kerül.

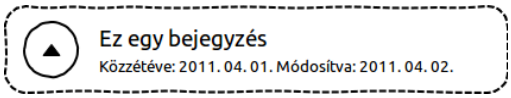
4.2.4. A bejegyzéslista

A bejegyzéslista a felhasználó által megírt összes bejegyzést tartalmazza. Vázlatos felépítése a 13. ábrán látható.



ábra 11. A bejegyzéslista

A bejegyzéslista az oldalsáv „Bejegyzéseim” feliratú fülén helyezkedik el. A főablak sablonjához képest két ponton találunk eltérést, a munkaterületen, valamint a műveletsávon. Ezt a két widgetet számmal jelöltem.



ábra 12. A bejegyzéslista egy eleme

Az 1-es számú panelen található a bejegyzések listája. A listában időrendi sorrendben szerepel az összes bejegyzés, amely valaha is keletkezett az adott blogon belül, kivételt csak a törölt és a lomtárban visszaállításra kijelölt bejegyzések képeznek (ezek a bejegyzések a „Lomtár” fül hasonló listájában kapnak helyet). A listaelemek szövege mindig a hozzájuk tartozó bejegyzés címe. Bár a fenti ábrán nem látszik, az egyes listaelemek mellett kis ábrák is lesznek, amelyek a bejegyzés pillanatnyi állapotát jelzik, valamint a bejegyzések címe alatt a bejegyzés létrehozásának és utolsó módosításának időpontja is szerepel. Vagyis egy konkrét listaelem úgy fog kinézni, ahogy az a 14. ábrán látható.

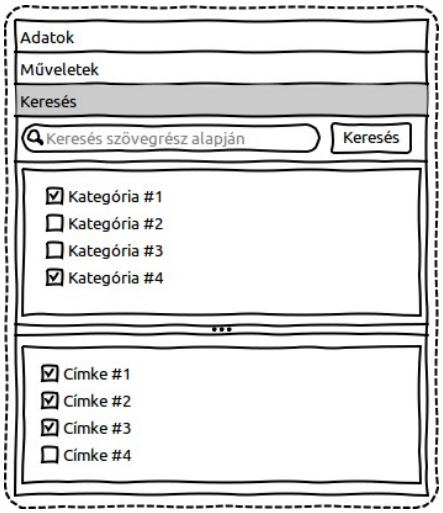
A 2-es számú panel, vagyis a műveletsáv tartalma az általuk végzett feladat alapján több csoportra is bontható. A csoportosítást egy különleges komponenssel, az ún. toolbox-szal szeretném megoldani.

A toolbox első „Adatok” csoportja a bejegyzés legfontosabb adatait jeleníti meg, vagyis a bejegyzés címét, státuszát, a létrehozás és a módosítás időpontját, esetleg a tartalom első pár sorát. A panel tartalma akkor frissül, ha a bejegyzéslistában valamelyik bejegyzésre kattint a felhasználó.

A második csoport, amely a „Műveletek” nevet viseli, azokat az opciókat tartalmazza, amelyeket egy, a bejegyzéslistában kijelölt bejegyzés esetében alkalmazhatunk. Hasonlóan

az „Adatok” panelhez, e panel tartalma is folyamatosan változik, attól függően, hogy a kijelölt bejegyzés milyen állapottal rendelkezik. Például egy, még be nem küldött piszkozat esetén csak olyan funkciók szükségesek, mint a szerkesztés, a beküldés, vagy az azonnali törlés. Az újraküldésnek, vagy a távoli blogból való törlésnek értelemszerűen ebben a helyzetben nincs értelme.

A harmadik csoport neve „Keresés”. A bejegyzéslista egy idő után terjedelmessé, és ezzel együtt kezelhetetlenné válhat, a rengeteg bejegyzés miatt. Szükséges dolog tehát, hogy a lista tartalmát különböző szempontok szerint szűkíthesse a program használója. Ezt a „Keresés” panel segítségével lehet megtenni.



ábra 13. Keresés a bejegyzéslistában

A „Keresés” panel felépítése a 15. ábrán látható.

A felső beviteli mezőben szövegrészletre lehet keresni, vagyis a programnak azt kell majd megvizsgálnia, hogy melyek azok a bejegyzések, amelyek tartalmában szerepel a megadott kifejezés.

Alatta a blog adatbázisában fellelhető összes kategória és címke található. A bejegyzések listájába csak azok a bejegyzések kerülhetnek, amelyekhez hozzá vannak rendelve a kijelölt kategóriák és címkék.

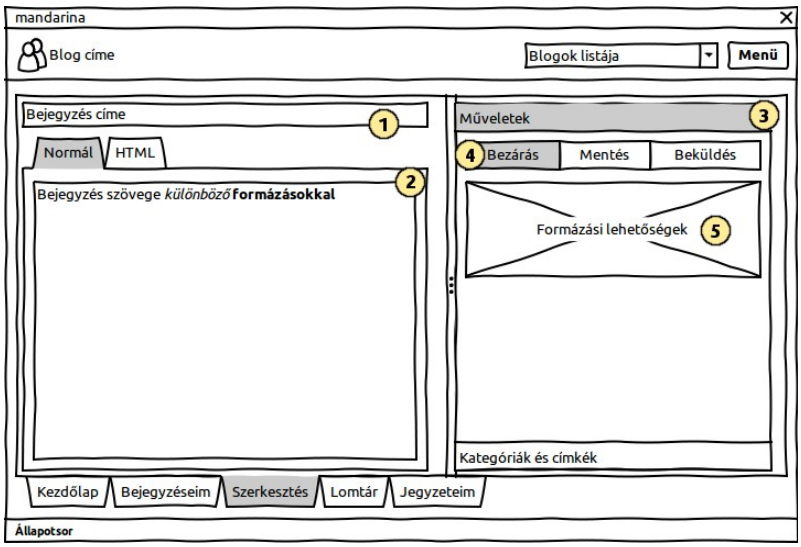
A bejegyzéslista tartalmának szűkítése a „Keresés” gombra való kattintással indulhat.

4.2.5. A lomtár

A „Lomtár” fül megjelenése hasonló a bejegyzéslistáéhoz. Alapvető különbség, hogy a listában csak a törölt, valamint a visszaállításra kijelölt bejegyzések jelennek meg, illetve a műveletsáv csak a végleges törléshez, valamint a visszaállításhoz kapcsolódó opciókat tartalmaz, vagyis nincs külön „Adatok” és „Keresés” csoport.

4.2.6. A bejegyzésszerkesztő

A mandarina program bejegyzésszerkesztője talán a felhasználói felület legösszetettebb része. Feladata, hogy egy könnyen kezelhető szövegszerkesztő-szerűséget biztosítson a bejegyzések tartalmának szerkesztéséhez. A bejegyzésszerkesztő vázlatát a 16. ábrán szemléltetem.



ábra 14. A bejegyzésszerkesztő

Azokat a widgeteket, amelyek említésre érdemesek, most is számozással jelöltem.

Az 1-es számmal jelölt beviteli mező a bejegyzés címét tárolja.

A 2-es számú widget a szerkesztőmező, ahol a bejegyzés tartalmát szerkesztheti a felhasználó. Két nézetet támogat: a „Normál” nézet egy WYSIWYG szerkesztőt tartalmaz, a „HTML” nézet pedig egy HTML nyelvet támogató szerkesztőt, a HTML nyelv szerelmeseinek. A két szerkesztőmező tartalmának szinkronizációja automatikusan meg kell, hogy történjen.

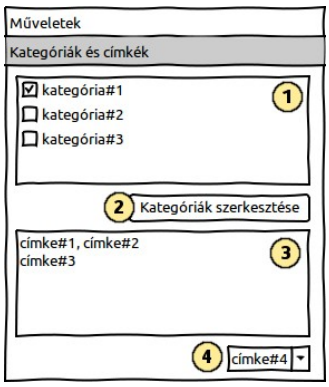
A műveletsávot ismét több részre kell bontanom, az áttekinthetőség érdekében, ehhez ismét egy toolboxot fogok használni, ez az ábrán a 3-as jelű felületi elem.

A 3-as toolbox 4-es jelű eleme egy három elemű gombsor. A Bezárás gombra kattintva a bejegyzés bezáródik, a változtatások elvetésével. A Mentés gombbal a bejegyzés új tartalma elmenthető az adatbázisba. A „Beküldés” gombbal pedig az újonnan létrehozott (vagy módosított) bejegyzés közvetlenül elküldhető a távoli blogmotor számára. Az ábrán nem látszik, de ez a nyomógomb egyben legördülő menü is lehet; ha a felhasználó nyomva tartja, a megjelenő menü segítségével piszkozatként is beküldheti a bejegyzést.

Az 5-ös jelű mező a formázó eszköztárt foglalja magába. Ide kerülnek azok a nyomógombok, amelyekkel a szövegszerkesztőben kijelölt szövegrészlet formátumát lehet állítani. Nem kell DTP-szintű lehetőségekkel számolni, csupán a legalapvetőbb, leggyakrabban használt, és a HTML nyelv által támogatott formázási opciókkal, mint például a félkövér, a dőlt, vagy az aláhúzott betűstílus, a bekezdések formátuma, vagy igazítása. Ezeken kívül olyan nyomógombok is megjelenhetnek, melyekkel egyéb fontos műveleteket lehet elvé-

gezni, mint például linkek beillesztése, képek, videók beágyazása.

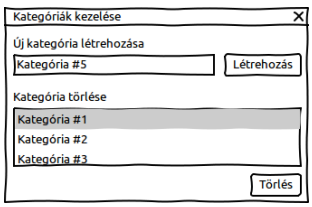
A 3-as számmal jelölt toolbox másik csoportja a „Kategóriák és címkék” csoport, melynek vázlata a 17. ábrán látható.



ábra 15. Kategóriák és címkék

Az 1-es számmal jelölt widget a blog adatbázisában tárolt összes kategóriát tartalmazza. Ha a készülő bejegyzéshez a felhasználó valamelyik kategóriát hozzá kívánja rendelni, nem kell mást tennie, mint a listában a megfelelő kategória neve előtti pipát bekapcsolni. A 2-es jelű gombra kattintva előbukkan a kategória-szerkesztő panel, ahol új kategóriákat lehet létrehozni, illetve régebbieket törölni.

A 3-as jelű szövegbeviteli mezőben kell felsorolni a bejegyzéshez tartozó címkéket. A címkék elválasztása soremeléssel, vesszővel, vagy pontosvesszővel történik. A 4-es számú legördülő listából a blog adatbázisban már korábban tárolt címkék közül lehet választani.



ábra 16. A kategória-szerkesztő

A 18. ábrán látható a fentebb említett kategória-szerkesztő panel.

Egy kategória létrehozásakor először be kell írni a beviteli mezőbe a kategória nevét (szövegét), majd a „Létrehozás” feliratú gombra kell kattintani.

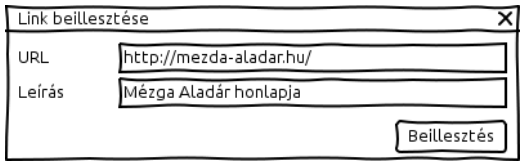
Kategória törlésekor a listából ki kell választani a törlendő kategóriát, majd a „Törlés” címkével ellátott gombot kell megnyomni.

4.2.7. Linkek beillesztése

A különböző internetes oldalakról származó linkeket kétféleképpen is be lehet illeszteni. A bejegyzésszerkesztő „HTML” nézetében az <a> tag paramétereként igen egyszerű. A „Normál” nézetben viszont nem biztos, hogy a szövegszerkesztő widget hasonlóképpen el fogja fogadni. Ezért a linkek beszúrásához külön felületet kell biztosítani.

A normál linkek esetén a 19. ábrán látható párbeszédpanel kerülhet elő ebben az esetben. A panelt a formázó eszköztár segítségével lehet előcsalogatni.

A felső szövegbeviteli mezőbe az internetes hivatkozás címe, az alsóba pedig a felhasználó által megadott rövid leírás kerül. A link a Beillesztés gombra való kattintással mindkét szö-



ábra 17. Link beillesztése

vegszerkesztő komponensbe bekerül.

Léteznek olyan internetes linkek is, amelyeket nem lehet ilyen egyszerűen elintézni. Ezek az úgynevezett beágyazott objektumok, amelyek struktúrája bonyolultabb, mint egy hivatkozásnak. Egy ilyen linket a 20. ábrán látható panellel lehet beilleszteni.

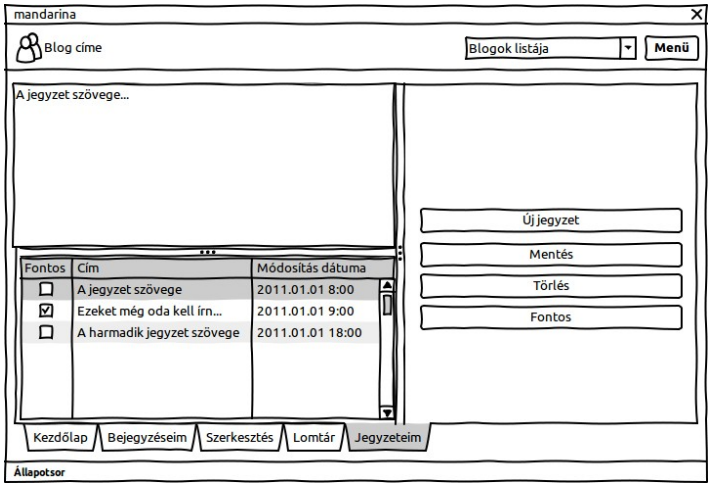
A beágyazáshoz szükséges HTML kódot a szövegbeviteli mezőben kell elhelyezni. A link beillesztése ezután a „Beágyazás” gombra való kattintással történik meg.

ábra 18. Kép beágyazása

A formázó eszköztáron két nyomógomb is van, amellyel ezt a panelt meg lehet nyitni, az egyik a képek, a másik pedig a videók számára van fenntartva. Ennek ellenére mindkét gomb ugyanezt a panelt nyitja meg. A program későbbi változatában ez a tervek szerint megváltozik, mindkét esetre más-más felületet fogok tervezni.

4.2.8. A jegyzettömb

A mandarina beépített jegyzettömbbel is rendelkezik. Ennek külalakját a 21. ábrán mutatom be.



ábra 19. A beépített jegyzettömb

A felhasználói felület természetesen itt is két részből áll. A bal oldali területen, vagyis a munkaterületen helyezkedik el a jegyzetek szerkesztőmezője. A módosítani kívánt jegyzetet az alsó táblázatból választhatja ki a felhasználó. Ezután a felette lévő szövegbeviteli mezőben lehet a jegyzet új tartalmát begépelni.

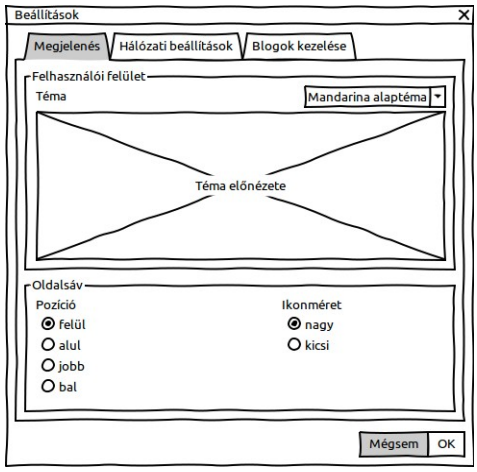
A műveletsávon négy gomb található, melyekkel a következő műveletek végezhetők el:

- új jegyzet létrehozása
- a módosított jegyzet tartalmának mentése
- létező jegyzet törlése
- jegyzet megjelelölése fontosként

4.2.9. A beállítások párbeszédpanel

A „Beállítások” nevű párbeszédablakot a főmenü gombra kattintva, vagy azt nyomva tartva, a „Beállítások...” menüparanccsal lehet előcsalogatni. A párbeszédpanel három fülből áll, ezek címkéje rendre „Megjelenés”, „Hálózati beállítások” és „Blogok kezelése”.

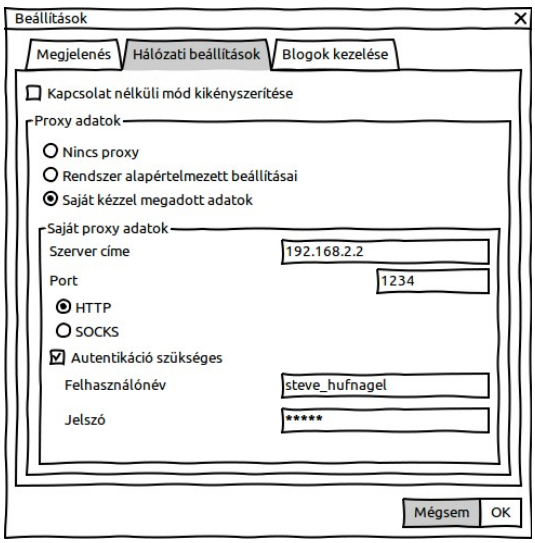
A „Megjelenés” lap vázlata látható a jobb oldali, 22. ábrán. A lap két részből áll.



ábra 20. Beállítások - Megjelenés

A felső részen a felhasználói felület megjelenését lehet beállítani. Szeretném, ha a program idővel támogatná a témákat, ez a csoport egyelőre tehát „helyfoglalónak” tekinthető. A legördülő panelen kiválasztott téma előnézete a „Téma előnézete” címkével jelölt területen fog megjelenni.

Az alsó, „Oldalsáv” nevű csoportban az oldalsávhoz kapcsolódó beállításokat lehet módosítani; a „Pozíció” feliratú csoportban a sáv elhelyezkedését (felül, alul, jobb, vagy bal oldalon), az „Ikonméret” csoportban pedig a sávon látható ikonok méretét.



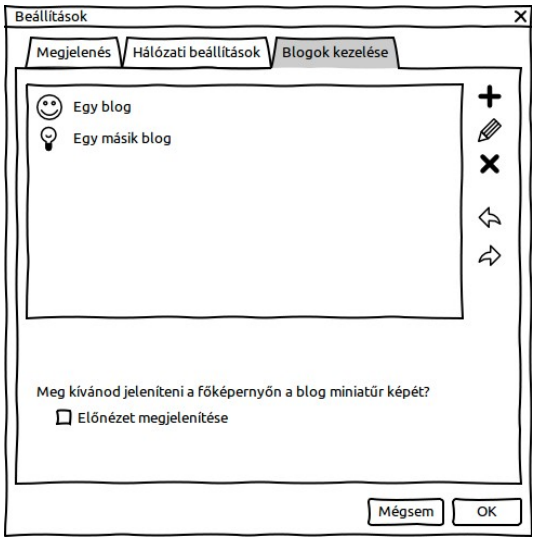
ábra 21. Beállítások - Hálózati kapcsolat

A „Hálózati beállítások” lap két nagyon fontos beállításnak ad helyet, ezek a kapcsolat nélküli munka kikényszerítéséhez, illetve a proxy-szerver adataihoz kapcsolódnak.

A „Kapcsolat nélküli mód kikényszerítése” feliratú checkbox kijelölésével a program már az indításakor alapértelmezés szerint kapcsolat nélküli módban indul.

A „Proxy adatok” csoportban a proxy-szerver konfigurálható. Említést érdemel a „Saját kézzel megadott adatok” rádiógomb. A „Saját proxy adatok” csoport elemei csak akkor szerkeszthetők, ha ez aktív, ellenkező esetben az egész csoport letiltásra kerül. Hasonlóképpen a felhasználónév és a jelszó is csak akkor adható meg, ha az „Autentikáció szükséges” jelölőnégyzet aktív.

A párbeszédablak harmadik lapja, melynek címe „Blogok kezelése”, a különböző blogok beállításait tartalmazza.



ábra 22. Beállítások - Blogok kezelése

A felső listában a programba regisztrált blogok listája látható. Az egyes listaelemek külalakja pontosan ugyanolyan, mint a főablak bloglistájának elemeié. A lista mellett található egy gombsor, ennek elemei a következő műveleteket jelentik:

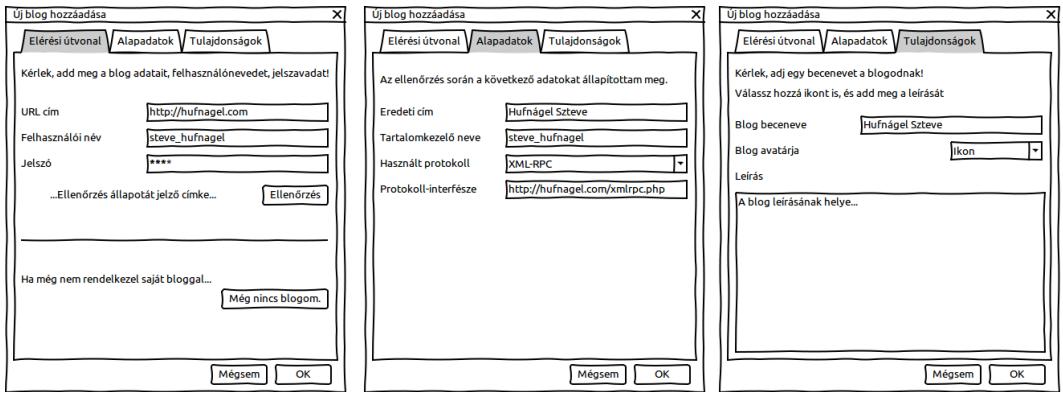
- új blog regisztrálása
- kijelölt blog adatainak módosítása
- kijelölt blog törlése
- importálás
- exportálás

Az első gombra való kattintáskor az „Új blog hozzáadása” nevű párbeszédablak, míg a második gombbal a „Blog adatainak módosítása” nevű párbeszédpanel indítható.

Az exportálás nyomógombbal a kijelölt bloghoz tartozó blog adatbázisból készíthetünk egy másodpéldányt, míg az importálás gombbal egy ilyen másodpéldányt állíthatunk vissza.

4.2.10. Új blog hozzáadása

A „Beállítások” párbeszédablakon található „Új blog hozzáadása” gombra való kattintás hatására jelenik meg az a párbeszédpanel, amely a 25. ábrán található.



ábra 23. Új blog hozzáadása

Az ablak első, „Elérési útvonal” nevű fülén adhatóak meg a blog eléréséhez szükséges legalapvetőbb adatok, ezek a blog URL címe, illetve a blogba való belépéshez szükséges felhasználói név és jelszó. Az „Ellenőrzés” gombra kattintva elindul a megadott adatok ellenőrzése, valamint a blog további adatainak kiderítése. Ennek állapotáról a gomb melletti szövegcímke tájékoztatja a felhasználót, ez a címke alapesetben láthatatlan.

Arra az esetre is gondolni kell, ha a felhasználónak még nincs saját blogja. Ilyenkor biztosítani kell számára egy felületet, ahonnan pár kattintással elérheti azokat a blogszolgáltatókat, akik szolgáltatását támogatja a mandarina. Ilyen szolgáltatók például a Wordpress.com, vagy a magyar Blogolj.net.

A második fül, amely az „Alapadatok” nevet viseli, azokat az adatokat tartalmazza, amelyeket a program az első lapon megadott adatok alapján ki tudott deríteni. Ezek a blog eredeti címe, a blogmotor neve, a publikáláshoz használt protokoll, illetve ennek a protokollnak az interfésze (ezzel az interfésszel tud a program kommunikálni a későbbiekben). Ezek a mezők szerkeszthetőek is, ehhez viszont az kell, hogy a felhasználó maximálisan biztos legyen a dolgában.

A harmadik fülön azok az adatok szerepelnek, amelyeket a felhasználónak kell mindenképpen megadnia, ugyanis a programban majd ezen adatok alapján tudja a blogot beazonosítani. Ezek az adatok a következők:

- a blog beceneve, amely lehet az eredeti cím, de lehet más is
- a blog avatárja
- egy opcionális leírás

Az első kettő adat mindig látható, a főablakban a blogfejlécben, a bloglistában, valamint a Beállítások panel megfelelő fülén. A leírás csak a kezdőoldalon jelenik meg.

4.2.11. Blog adatainak szerkesztése

A felvett blog adatai utólag is szerkeszthetők, az ehhez készített felhasználói felület a 26. ábrán látható.

ábra 24. Blog adatainak szerkesztése

Az első, „Alapadatok” lapon a felhasználó által „büntetlenül” módosítható adatok szerepelnek, vagyis a blog beceneve, avatárja és a rövid leírás.

A második lapon (ennek neve „Haladó”) azok a beállítások találhatóak, amelyeket a program használ a blogmotor eléréséhez. Amikor azt merészeltem írni, hogy az alapadatok „büntetlenül” módosíthatóak, akkor arra gondoltam, hogy azok az adatok csak a programon belüli belső azonosíthatóságot szolgálják, a program szempontjából igazából nincs jelentőségük. Ezzel szemben a haladó beállításoknak annál inkább, a felhasználó által helytelenül megadott adatok azt eredményezhetik, hogy a program nem fog megfelelően működni. Tehát ezen adatok módosítása körültekintést igényel a felhasználó részéről. Ezért is bontottam két részre a blog beállításait.

A harmadik, „Képességek” címkéjű lapon a blogmotor képességei láthatóak. Ezeket az adatokat a blog regisztrációjakor a program deríti ki, az „Új blog hozzáadása” panelen megadott adatok alapján. Magammal folytatott hosszas gondolati tanácskozás után arra az eredményre jutottam, hogy ezek az adatok nagyon ritkán változnak meg, így ezek utólag nem módosíthatóak.

5. A BlogDB adatbázis

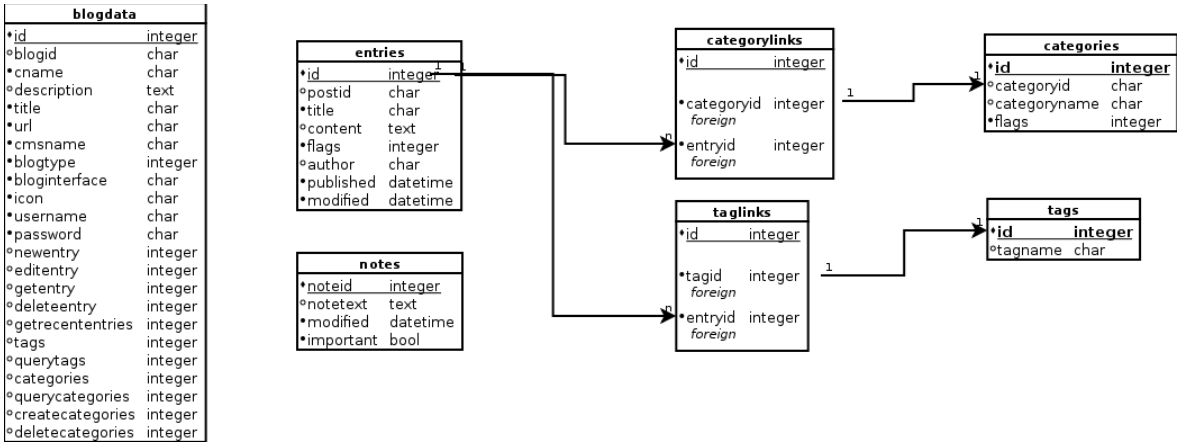
A mandarina program a blogból letöltött, illetve a helyileg tárolt bejegyzéseket egy relációs adatbázisban tárolja. (Erre az adatbázisra a későbbiekben mBlogDB néven fogok hivatkozni.) Ennek az az oka, hogy az adatbázisban sokkal egyszerűbb a bejegyzések és a hozzájuk tartozó egyéb adatok (címkék, kategóriák, hozzászólások, jegyzetek) tárolása, mint teszem azt, egy szövegfájlban vagy egy XML fájlban, illetve az adatok visszanyerése is hasonlóan egyszerű. Az adatbázis motorjaként a szabad és nyílt forráskódú SQLite 3[20]-at fogom alkalmazni, egyszerűsége és gyorsasága miatt.

5.1. A BlogDB felépítése

A BlogDB adatbázis elérési útvonala a felhasználó saját könyvtárában található. Ennek helye Linux rendszer esetében ~/.mandarina/blogs/ mappában, újabb Windows rendszereken (Vista, 7) az %UserProfile%\.mandarina\blogs\ könyvtárban található. Minden egyes beállított bloghoz külön BlogDB adatbázis tartozik.

A fájl neve hat darab véletlenszerű számból áll, kiterjesztése *.db, az SQLite természetéből fakadóan ez lesz az adatbázis neve.

Az adatbázis tábláit a 27. ábra mutatja.



ábra 25. A BlogDB felépítése

5.1.1. Blogdata tábla

A blogdata tábla a blog adatait tartalmazza.

Az id mező a tábla elsődleges kulcsa.

A blogname mező a blog címe, tulajdonképpen az a rövid leírás, amely egy webböngésző ablakának címsorában is megjelenne.

20 http://www.sqlite.org/

A cname mező a blog úgynevezett beceneve, amelyet a felhasználó ad meg a blog programba történő regisztrálásakor.

A blogicon mező a bloghoz tartozó ikon elérési útvonalát tartalmazza. Ezt az ikont a felhasználó adja meg a blog azonosítására, és a főablak legfelső részén, a blog beceneve előttt jelenik meg.

A blogtype mező a blog alapjául szolgáló tartalomkezelő elnevezése, illetve, ha ez kideríthető, a verziószáma.

Az username és a password mező a felhasználó azonosításában játszik igen fontos szerepet, előbbi a belépéshez használt felhasználónév, utóbbi a jelszó.

A blogurl mező a blog URL címe.

A description mező egy rövid leírás a blogról, a felhasználó adja meg, a későbbi verziókban lesz jelentősége.

A bloginterface mező a blog által ismert, és a felhasználó által kijelölt blogprotokoll interfészének címe. A mandarina program erre az URL címre küldi a protokollnak megfelelő kéréseket és innen fogadja a válaszokat is. Mivel a program jelenleg csak az XML-RPC blogprotokollt támogatja, ez minden esetben az XML-RPC interfész URL címe, általában a blog trackback címe.

A blogid mező a blog belső azonosítója, az XML-RPC protokoll megfelelő működéséhez szükséges.

Az ezután következő mezők a blog motorjának képességeit írják le. Mindegyik mező logikai érték, típusuk csak azért integer, mert az SQLite adatbázis-kezelő nem ismeri a boolean típust [25]. Ezekre a mezőkre azért van szükség, hogy a mandarina bizonyos szolgáltatásokat (pl. kategóriák kezelése, címkék kezelése) letilthasson, ha azokat a blogmotor nem támogatja. Bár a mezők nevei egyértelműek, most mégis ismertetném a jelentésüket.

A newentry, editentry, getentry, deleteentry mezők rendre a következő képességeket jelentik: új bejegyzés létrehozása, létező bejegyzés módosítása, létező bejegyzés lekérése, illetve egy bejegyzés törlése.

A getrecententries értéke mutatja meg, hogy a blogmotor képes-e a legutolsó bejegyzések kötegelt lekérdezésére.

A tags, és a querytags mezők a címkék támogatásához kapcsolódnak. A tags mező értéke megmutatja, hogy a blogmotor egyáltalán támogatja-e a címkéket, a querytags mező értéke

pedig, hogy a blogmotor képes-e az összes címkét lekérdezni.

A categories, a querycategories, a createcategories és a deletecategories mezők a kategóriák kezelésénél játszanak szerepet. Az első két mező (categories, querycategories) szerepe hasonló, mint a tags és querytags mezőknek. Az XML-RPC protokoll egyik sajátossága, hogy nem minden esetben tartozik a blogmotor minden tulajdonságához egy XML-RPC metódus, így előfordulhat, hogy a blogmotor támogatja a kategóriákat, de XML-RPC-n keresztül csak lekérni tudjuk a kategóriákat, létrehozni és törölni viszont nem. Ezen metódusok létezését a createcategories és a deletecategories értéke mutatja meg.

5.1.2. Entries tábla

Az entries tábla a bloghoz tartozó bejegyzéseket foglalja magában. A program itt tárolja a blogból szinkronizált bejegyzéseket, a beküldésre váró helyi piszkozatokat, a blogból már törölt, de a helyi szemetesben még benne lévő bejegyzéseket.

Az egyes mezők ismertetése:

Az id a tábla elsődleges kulcsa, a később ismertetésre kerülő MBlogDB osztály bejegyzéskezelő metódusai ezen azonosító segítségével azonosítják be a kívánt bejegyzést.

A postid mező a bejegyzés egyedi belső azonosítója, ezen azonosító segítségével azonosítja a blogmotor a bejegyzést.

A title a bejegyzés címe, a content pedig a bejegyzés teljes szövege. Utóbbi HTML kódot is tartalmazhat, így szerepelhetnek benne olyan karakterek is, amelyek miatt az SQL értelmező „megzavarodhat” (ilyen karakterek lehetnek az idézőjel, a tag-ekben használt relációs jelek), így ezeket majd a megfelelő módon konvertálni kell. (Kézenfekvő módszer például a HTML escape karaktereinek használata)

A flags mező a bejegyzés állapotát rögzíti. Mivel a mandarina internetkapcsolat nélkül is működőképes kell, hogy legyen, valahogyan rögzíteni kell, hogy az egyes bejegyzésekkel mit kell tennie a programnak, ha a felhasználó ismét online tud lenni. Meg kell határozni, melyek azok a bejegyzések, amelyeket újként kell elküldeni a blogmotor számára, melyek azok, amelyeket módosítani, vagy netán törölni kell. Erre egy megoldás a flagek használata, amely egy „ősi” programozási technika. A mező típusa integer, az egyes állapotok is integer típusú számok, mégpedig 2 egész számú többszörösei. Az állapotok beállítása aritmetikai (összeadás és kivonás), lekérdezése pedig logikai műveletekkel (AND) történhet. Az egyes flageket a megfelelő helyen ismertetni fogom <<hivatkozás>>.

Az author a bejegyzés szerzőjének azonosítója, a program egy későbbi változatában lesz szerepe.

A published és a modified mező a bejegyzés keletkezésének és utolsó módosításának idejét tárolja. A dátum tárolásának formátuma az SQL szabvány dátum formátuma.

5.1.3. Notes tábla

A notes tábla a bloggal kapcsolatos felhasználói feljegyzéseket, jegyzeteket tárolja. Ezek csak a programon belül jelennek meg, csak itt szerkeszthetők. Tartalmuk soha, semmilyen körülmények között nem kerül feltöltésre a blogba.

A noteid mező a jegyzet programon belüli azonosítását szolgálja.

A notetext a jegyzet szövegét, a datetime mező a legutolsó szerkesztés időpontját tartalmazza.

Az important mező a bejegyzés fontosságát jelöli meg. Értéke 0, ha a jegyzet nincs megjelölve, illetve egy 0-nál nagyobb pozitív egész szám (általában 1), ha a jegyzet fontosként lett megjelölve.

5.1.4. Tags tábla

A tags tábla a program által nyilvántartott összes címkét (tag) tartalmazza. A táblában szereplő címkék között a távoli blog által nyilvántartott címkék mellett azok is szerepelnek, amelyek ugyan hozzá lettek rendelve egy bejegyzéshez, de mivel az eképpen módosított bejegyzések változtatásai még nem lettek elküldve, még nem jöttek létre.

A tábla végtelenül egyszerű, csupán két mezőt tartalmaz. Az id mező a tábla elsődleges kulcsa, a tagname mező pedig a címke szövege.

5.1.5. Taglinks tábla

A taglinks tábla feladata az egyes bejegyzések és a hozzájuk tartozó címkék kapcsolatának nyilvántartása. Az „összekapcsoláshoz” két mező szükséges, az adott címkéhez tartozó elsődleges kulcs (tagid), illetve az adott bejegyzéshez tartozó elsődleges kulcs(entryid), mint idegen kulcsok.

5.1.6. Categories tábla

A categories tábla szerepe hasonló, mint a tags tábláé, de értelemszerűen a tábla a blog kategóriáit tartalmazza. Az XML-RPC-alapú blogmotorokban egy bejegyzéshez csak létező kategóriákat lehet hozzárendelni, ezért egy bejegyzés beküldése előtt a kívánt kategóriákat

mindenképpen létre kell hozni. Ennek következménye, hogy a categories táblában is hasonló állapotjelző rendszert kell kidolgozni, mint az entries táblában.

Az id mező a tábla elsődleges kulcsa. A categoryid a blogmotor által használt kategória-azonosító, az XML-RPC protokollal való kompatibilitásért szükséges. A categoryname a kategória szövege. A flags mező szerepe ugyanaz, mint az entries tábla azonos nevű mezőjének, ez az a bizonyos állapotjelző rendszer, ami a program offline módban történő működéséhez szükséges. Az egyes kategória flageket a megfelelő helyen ismertetem.

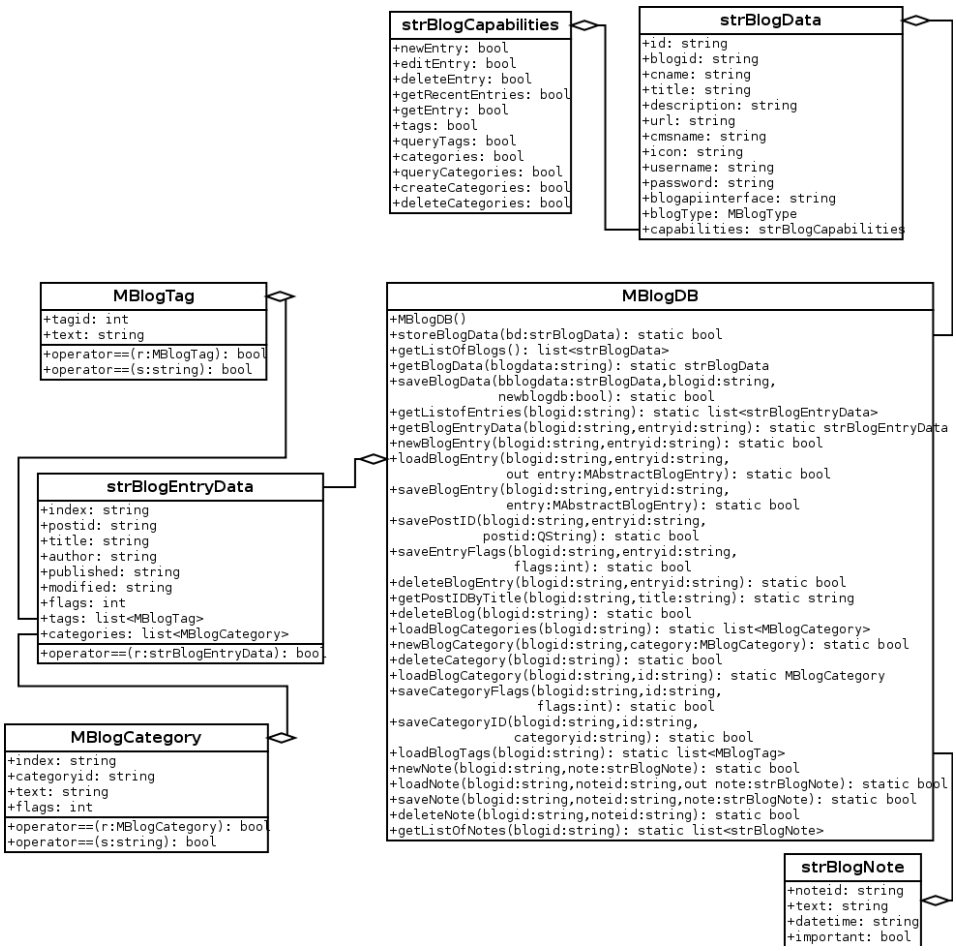
5.1.7. Categorylinks tábla

A categorylinks tábla feladata szintén hasonló a taglinks táblához, itt azonban a kategóriák és a bejegyzések kapcsolatát definiálhatjuk. A kötések létrehozásának elve is teljesen azonos, az adott kategóriához tartozó elsődleges kulcs (categoryid), illetve az adott bejegyzéshez tartozó elsődleges kulcs(entryid) hordozza a kapcsolatot.

5.2. Az mBlogDB osztály

Az mBlogDB osztály feladata a program és a BlogDB közötti kapcsolat fenntartása. Pusztán public hozzáférésű, statikus metódusokból áll. Célja, hogy a blogprotokoll kezelésének folyamatát leíró MabstractBlog osztály kódját különválasszam a BlogDB kezelését végző kódtól. Így lehetővé válik, hogy a későbbiekben az SQLite 3 helyett esetleg más adatbázis (vagy eltérő SQLite verzió) kezelésére is felkészítsük a programot, hiszen csak az mBlogDB osztály kódját kell kicserélni.

A következő ábrán az mBlogDB osztály diagramja látható.



ábra 26. Az MBlogDB osztálydiagramja

5.3. Adattípusok

Az MBlogDB osztály a következő „saját készítésű” adattípusokat használja. Ezen adattípusok közül néhányan más osztályoknál is feltűnnek majd, mint például az MAbstractBlog, MAbstractBlogEntry és ezek leszármazott osztályai.

5.3.1. StrBlogData

Az strBlogData struktúra egy programban regisztrált blog alapadatait tartalmazza. Minden egyes BlogDB adatbázis leképezhető egy strBlogData struktúrára. A struktúrában egy másik, beágyazott struktúra is található.

Az egyes mezők szerepe a következő:

- id: azon BlogDB fájlneve, amelynek adatait a struktúra tartalmazza
- blogid: a távoli blog azonosítója, amelyet a protokoll használ a kérések elküldésénél
- cname: a távoli blog beceneve, amelyet a felhasználó adhat meg

- title: a távoli blog eredeti címe,
- description: egy leírás, amelyet szintén a felhasználó adhat meg (bár nem szükséges)
- url: a blog URL címe
- cmsname: a blog motorjának neve, ez lehet például „Wordpress”, „Drupal”, „Blogger”, vagy ha nem állapítható meg, akkor „Ismeretlen”
- icon: a blog avatárja (ikonja), a felhasználó adja meg
- username és password: a felhasználó neve és jelszava, amit a blogba történő bejelentkezéskor is használ. Az autentikációt igénylő műveleteknél szükségesek.
- blogapiinterface: a blogprotokoll interfésze
- blogType: enum, a blogprotokoll típusa, a program jelenlegi változatában értéke fix, btXmlRpc
- capabilities: struktúrába ágyazott struktúra, a blogmotor képességeit írja le. Ismertetését lásd később

5.3.2. StrBlogEntryData

Az strBlogEntryData struktúra egy blogon belüli bejegyzést reprezentál. Minden fontosabb adatot tartalmaz, egyetlen kivétellel. Bizonyos megfontolásból a bejegyzések szövege nem töltődik be, ezzel csökkentve egy bejegyzés memóriafoglalásának mértékét.

A struktúra egyes mezőinek jelentése a következő:

- index: az adatbázisban a bejegyzéshez tartozó elsődleges kulcs, a bejegyzés későbbi, programon belüli azonosításához kell
- postid: a bejegyzés blogprotokollon belüli azonosítója
- title: a bejegyzés címe
- author: a bejegyzés szerzőjének azonosítója
- published: a bejegyzés keletkezésének ideje
- modified: a bejegyzés legutolsó módosításának ideje
- flags: a bejegyzés állapotát jelző flag

- tags: a bejegyzéshez tartozó címkék listája
- categories: a bejegyzéshez tartozó kategóriák listája

5.3.3. MBlogTag

Az MBlogTag struktúra egy címkének (tag) felel meg. Nemcsak az MBlogDB osztály, hanem az MabstractBlog, MAbstractBlogEntry és ezek leszármazottai is használják.

A struktúrában a következő mezők találhatók:

- tagid: az adatbázisban tárolt címke elsődleges kulcsa, a programon belüli azonosítás céljából szerepel
- text: a címke szövege

A struktúra két metódust is tartalmaz:

- operator==(): a C++ „==” operátorának átdefiniálása azért szükséges, hogy könnyebben el lehessen két címkéről dönteni, hogy azonosak-e. Az azonosság feltétele, hogy a címkék szövege azonos legyen. A függvénynek két alakja van, az egyik alak egy MblogTag-et, a másik pedig egy egyszerű sztringet vár paraméterként.

5.3.4. MblogCategory

Az MblogCategory struktúra az MblogTag párja, abban a tekintetben, hogy a két struktúra felépítése hasonlít egymásra. Az egyedüli különbség (amellett természetesen, hogy a struktúra egy kategóriát jelképez, nem pedig egy címkét) a mezők száma.

A struktúra felépítése:

- index: az adatbázisban szereplő kategória elsődleges indexe, a programon belüli azonosítást szolgálja
- categoryid: a kategória blogprotokollon belüli azonosítója
- text: a kategória neve
- flags: a kategória állapotát jelzi. Céljának bemutatását lásd a BlogDB categories táblájának ismertetésénél...

5.3.5. StrBlogCapabilities

Az strBlogCapabilities struktúra a blogmotor képességeit írja le, vagyis azt, hogy a mandarina által támogatott szolgáltatásokat (bejegyzések kezelése, kategóriák, címkék támoga-

tása) a blogmotor ismeri-e. A struktúra egy az egyben a BlogDB adatbázis blogdata táblájának megfelelő sorait képezi le.

5.3.6. StrBlogNote

Az strBlogNote struktúra egy bloghoz tartozó felhasználói jegyzetet tárol. A felhasználói jegyzet kizárólag a programon belül jelenik meg, a struktúra a mandarina beépített jegyzettömbjének működéséhez szükséges.

A struktúra mezői:

- noteid: a jegyzethez tartozó elsődleges kulcs, az adatbázison belüli azonosításhoz
- text: a jegyzet teljes szövege
- datetime: a jegyzet utolsó módosításának ideje
- important: logikai érték, amely a jegyzet fontosságát jelzi

5.4. Fontosabb algoritmusok

5.4.1. Bejegyzések betöltése

Az algoritmus feladata, hogy a BlogDB-ből egy bejegyzést a hozzá való kategóriákkal, címkékkel együtt betöltsön. Az algoritmus a megfelelő blogot a blogid alapján (az adatbázis neve, hat karakterből álló számsor), a megfelelő bejegyzést pedig a bejegyzés indexe alapján (entries tábla id nevű elsődleges kulcsa) azonosítja.

Az algoritmus ismertetéséhez először szeretnék néhány absztrakt adattípust definiálni.

StrEntryData : egy bejegyzést reprezentáló struktúra

strCategory : egy kategóriát reprezentáló struktúra

strTag: egy címkét reprezentáló struktúra

Database : egy adatbázis-kezelő osztály

Következzen maga az algoritmus...

```
// Bemenet
blogid : string;                // a blog adatbázis fájlneve
entryid: string;                // a betölteni kívánt bejegyzés id-je
entry : strEntryData            // a bejegyzés adatait tartalmazó struktúra

;adatbázis megnyitása
ha (a megnyitás sikeres volt) {
        ;SELECT * FROM entries WHERE id=entryid --> entry
```

```
        // kategóriák beolvasása
        ;select c.id, c.categoryname from categories c, categorylinks clink where clink.categoryid=c.id
        and clink.entryid=entryid --> entry.categories

        // címkék beolvasása
        ;select t.id, t.tagname from tags t, taglinks tlink where  tlink.tagid=t.id and
        tlink.entryid=entryid --> entry.tags
}
// Kimenet
entry, a blogDB-ből beolvasott bejegyzés adataival feltöltve
```

Az algoritmust az MBlogDB osztály loadEntry nevű statikus metódusa valósítja meg.

5.4.2. Bejegyzések mentése

Az algoritmus egy bejegyzést ment el a BlogDB-be. Először a bejegyzés alapadatait, majd a bejegyzés kategóriáit és címkéit menti el, végül pedig a megfelelő kategórialinkeket és címkelinkeket is létrehozza. A blog és a bejegyzés azonosítása a betöltés algoritmusban megismert módon történik: a blogot a hatjegyű blogid alapján, magát a bejegyzést pedig az indexe alapján.

Az algoritmus a következő adattípusokat használja:

StrEntryData : egy bejegyzést reprezentáló struktúra

strCategory : egy kategóriát reprezentáló struktúra

strTag: egy címkét reprezentáló struktúra

Database : egy adatbázis-kezelő osztály

Következzen maga az algoritmus...

```
// Bemenet
ed : strEntryData                          // az elmentendő bejegyzés
blogid : string                   // a blog azonosítója (az adatbázis neve)
blog.categories : <list> strCategory      // a blog kategóriái
blog.tags : <list> strTag         // a blog címkéi
entry.categories : <list>strCategory      // a bejegyzés kategóriái
entry.tags : <list>strTag         // a bejegyzés címkéi
;adatbázis megnyitása

// ha (sikerült az adatbázis megnyitása) {
        // bejegyzés alapadatainak mentése
        // ha a bejegyzés még új
        ha (ed.flags==flNew) {
                ;insert or replace into entries (postid, title, content, flags, author, published,
                modified) values (ed.postid, ed.title, ed.content, ed.flags, ed.author, ed.published,
                ed.modified
        }
        // ha a bejegyzés már létezik
        egyékként {
                ;update entries set postid=ed.postid, title=ed.title, content=ed.content,
                flags=ed.flags, author=ed.author, published=ed.published, modified=ed.modified
```

```
        }
        // kategóriák mentése
        // létező kategórialinkek törlése
        ;delete from categorylinks where entryid=ed.index”
        // új kategóriák mentése
        minden cat-re entry.categories-ben {
                ha (blog.categories nem tartalmazza cat-et)
                        ;insert into categories(categoryid, categoryname,  flags) values(cat.cate
                                goryid, cat.text, flNew);

        }
        // kategórialinkek létrehozása
        minden cat2-re entry.categories-ben {
                ;insert into categorylinks (categoryid, entryid) values(cat2.index, entry.index);
        }
        // címkék mentése
        // létező címkelinkek törlése
        ;delete from taglinks where entryid=ed.index”);
        // új címkék mentése
        minden tag-re entry.tags-ben {
                ha (blog.tags nem tartalmazza tag-et) {
                        ;insert into tags(tagname) values(tag.text);
                }
        }
        // címkelinkek létrehozása
        minden tag2-re entry.tags-ben {
                ;insert into taglinks (tagid, entryid) values(tag2.index,                   entry.index
        }
}
// Kimenet
        // nincs kimenet
```

Az algoritmust a gyakorlatban két részre bontottam. Az első esetet, vagyis az új bejegyzés létrehozását az MBlogDB osztály newEntry nevű statikus metódusa, a létező bejegyzések módosítását pedig az MBlogDB osztály saveEntry nevű statikus metódusa valósítja meg.

6. A blogkezelő osztályok

Az előző fejezetben megismerkedhettünk a program által kezelt blogprotokollal, az XML-RPC alapú publikációs protokollal, valamint az MBlogDB osztállyal, amely a blog adatbázisához nyújt egy könnyen kezelhető interfészt. A következőkben szeretném bemutatni azokat az osztályokat, amelyek ennek a két technológiának a lehetőségeit próbálja összefogni, ennek segítségével lehetővé tenni a bejegyzések, kategóriák, címkék teljes körű kezelését.

6.1. Az alapvető funkciók

A blogkezelő osztályok alapvető feladatai a következők:

- blogbejegyzések kezelése: ide több feladatot is besorolhatunk: bejegyzés betöltése,

mentése, elküldése a blogmotor számára, illetve egy létező bejegyzés módosítása, vagy törlése

- blog kategóriáinak és címkéinek kezelése: az itt felmerülő feladatok hasonlóak, mint a bejegyzések kezelésénél
- bejegyzések keresése különböző szempontok szerint
- a távoli blog és a blogadatbázis tartalmának szinkronizálása

6.2. A legfontosabb osztályok és típusok ismertetése

6.2.1. FlEntryState

Mint már korábban írtam, ahhoz, hogy az offline munkamenet biztosítható, illetve a szinkronizáció, mint olyan, gond nélkül végrehajtható legyen, szükség van az egyes bejegyzések állapotának rögzítésére, hogy tudjuk, melyek azok a bejegyzések, amelyek még csak helyi piszkozatok, melyek azok, amelyek már be lettek küldve, vagy melyek azok, amelyek esetleg törölve lettek. Lehetnek olyan bejegyzések, amelyekkel a felhasználó szeretett volna valamit csinálni, de nem tudott, mert a művelet valamilyen okból (például nem volt internetkapcsolat) félbeszakadt. Az ilyen eseteket is jelölnünk kell valahogyan. Erre szolgál az flEntryState enum típus, amellyel gyakorlatilag bináris kapcsolókat (flageket) definiálunk. Az állapot olvasása ezután bitenkénti logikai műveletekkel, módosítása pedig aritmetikai műveletekkel történhet meg.

Az flEntryState értékeit és jelentésüket a következő táblázat foglalja össze.

FlesNew=0	új bejegyzés
flesNormal=1	normál bejegyzés
flesDraft=2	piszkozat
flesLocalDraft=4	helyi piszkozat
flesMarkedForSubmit=8	beküldésre kijelölt bejegyzés
flesMarkedForModify=16	módosításra kijelölt bejegyzés
flesMarkedForDelete=32	törlésre kijelölt bejegyzés
flesMarkedForRecover=64	visszaállításra kijelölt bejegyzés
flesDeleted=128	törölt bejegyzés
flesSubmitted=256	beküldött bejegyzés

1. táblázat. Bejegyzések állapota és a hozzájuk tartozó értékek

6.2.2. FlCategoryState

Mivel a kategóriákat a használatuk előtt létre kell hozni, ehhez pedig protokollműveletek szükségesek, fontossá válik, hogy a kategóriák állapotát is valamiképpen jelöljük. Ez a bejegyzésállapotok jelöléséhez hasonló módon történhet. A megfelelő értékeket az flCategoryState enum típus határozza meg. A következő táblázat a flCategoryState lehetséges értékeit tartalmazza.

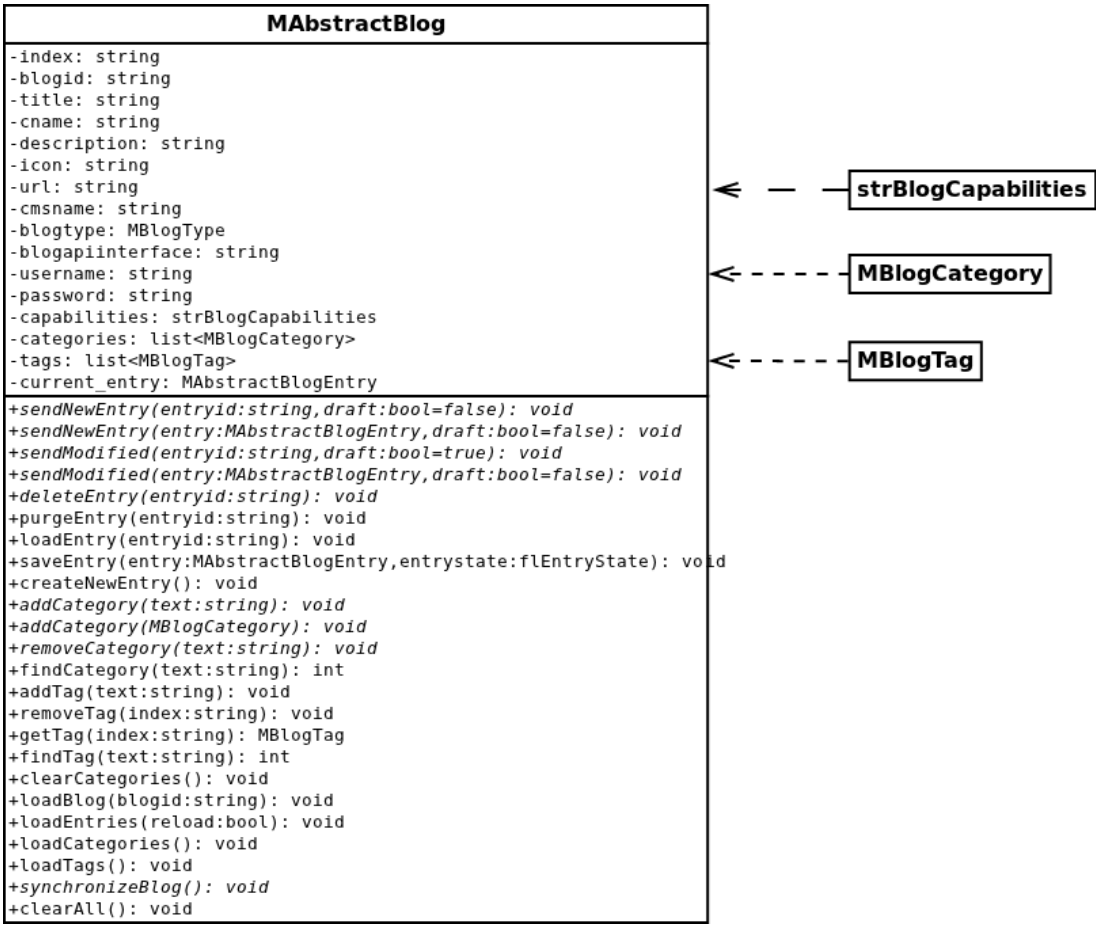
FlcsNew=0	új kategória
flcsNormal=1	még nem létező, de már nyilvántartott kategória
flcsMarkedForSubmit=2	beküldésre kijelölt kategória
flcsMarkedForDelete=4	törlésre kijelölt kategória
flcsDeleted=8	törölt kategória
flcsSubmitted=16	beküldött, vagyis létező kategória

2. táblázat. Kategóriák állapota és a hozzájuk tartozó értékek

6.2.3. MAbstractBlog

Talán a legfontosabb osztály mind közül. A különböző publikációs protokollokat kezelő osztályok közös őse. A blogkezelés alap mechanizmusait megvalósító tagfüggvényeken kí-

vül tartalmaz még pár virtuális tagfüggvényt, ezek jellemzően azok a tagfüggvények, amelyek viselkedése protokollfüggő.



27. ábra. Az MAbstractBlog diagramja

Az MAbstractBlog osztály diagramja látható a 29. ábrán.

Az osztálydiagram természetesen nem teljes. A private hozzáférésű adattagok lekérdező és beállító tagfüggvényeit kihagytam, ezek nélkül is épp elég impozáns méretű az ábra.

Az strBlogCapabilities, MBlogCategory és MBlogTag struktúrákat már ismerjük, a BlogDB-ről szóló fejezetből. Hasonlóképpen ismerősek lehetnek az MAbstractBlog osztály adattagjai, vagyis a cname, title, description, url, blogid, cmsname, icon, username, password, blogapiinterface, index, blogtype és capabilities mezők. Ezek ugyanazon célt szolgálják, mint a korábban ismertetett mBlogDB osztály esetén; a blogról nyilvántartott legfontosabb adatokat tartalmazzák. A categories és tags adattagok a blog adatbázisában szereplő összes kategóriát és címkét tartalmazzák.

A dőlt betűvel szedett metódusok virtuális metódusok, ezeket az osztály leszármazottjai fogják majd megvalósítani, a „rájuk bízott” blogprotokollnak megfelelően, így ezeket a tagfüggvényeket és működésüket később ismertetem.

A loadBlog() metódus a blog betöltéséért felel. A függvény a megadott azonosítójú blog adatbázisban tárolt adatok alapján feltölti az osztályból példányosított objektum adattagjait. A loadEntries(), loadCategories() és loadTags() metódusok rendre a bejegyzések, kategóriák és címkék betöltését, vagy újratöltését végzik el.

Ahhoz, hogy egy bejegyzést szerkeszteni tudjunk, az osztályt pár fontos metódussal el kell látni. A loadEntry(entryid:string) metódus a paraméterként megadott bejegyzést tölti be az osztály current_entry nevű adattagjába. Ennek párja a saveEntry(), amely a korábban betöltött bejegyzés megváltozott tartalmát visszaírja az adatbázisba. E két függvény algoritmusát később bemutatom. Új bejegyzés készítéséhez a createNewEntry() függvényt kell majd meghívnunk.

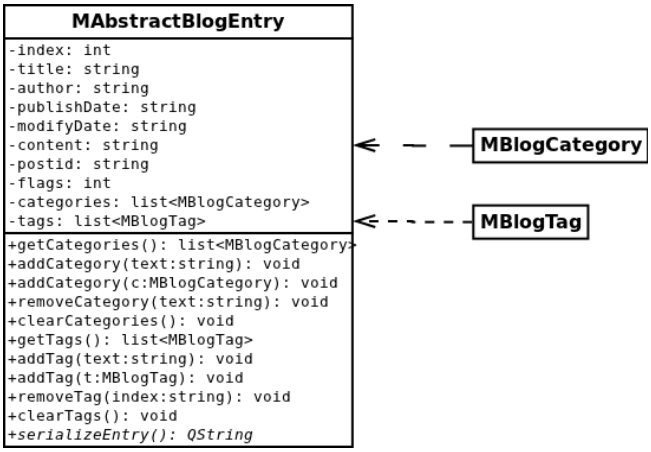
A kategóriák és címkék kezelését is ez az osztály végzi. A címkekezelő és a kategóriakezelő metódusok nagyon hasonlítanak egymásra, egyetlen lényegi különbség, hogy utóbbiak virtuális függvények, a bejegyzések beküldését, módosítását, törlését lehetővé tévő függvényekhez hasonlóan. Ennek oka a kategóriák és a címkék eltérő természetében keresendő. Úgy figyeltem meg, hogy egy kategóriát használat előtt létre kell hozni, míg a címkék esetén ez nem szükséges. A kategóriák létrehozása pedig protokollfüggő. A synchronizeBlog() függvénnyel elindul a távoli blog és a blogadatbázis tartalmának szinkronizálása.

Esetenként elképzelhető, hogy egy kategóriát vagy egy címkét az adatbázisból a neve alapján szeretnénk megkeresni. Erre szolgál a findCategory(text:string) és a findTag(text:string) függvény.

A blogkezelő objektumot a clearAll() függvénnyel lehet alaphelyzetbe állítani.

6.2.4. MAbstractBlogEntry

Ahogy az MAbstractBlog osztály egy blogot, az MAbstractBlogEntry egy bejegyzést reprezentál. Ez az osztály csak olyan eljárásokat tartalmaz, melyek protokollfüggetlenek. A protokollfüggő metódusokról a leszármazottak gondoskodnak majd.



28. ábra. MAbstractBlogEntry

Az osztály diagramja a 30. ábrán látható. Hasonlóan az MAbstractBloghoz tartozó diagramhoz, a private

adattagok lekérdező és beállító metódusait itt is kihagytam.

Az osztály adattagjainak szerepe megegyezik az strBlogEntryData osztály adattagjaival, de a content mezőben a bejegyzés teljes szövegét is eltárolom.

Az osztály metódusainak többsége a bejegyzéshez tartozó kategóriák és címkék kezelését teszik lehetővé. A getCategories() és a getTags() metódusokkal a bejegyzés kategóriáinak illetve címkéinek listáját kérhetjük le. Az addCategory() polimorf függvénnyel egy teljesen új, vagy már létező kategóriát adhatunk hozzá a bejegyzéshez, míg a removeCategory()-val egy korábban hozzáadott kategóriát törölhetünk. Ezek a metódusok a categories lista tartalmát manipulálják. A clearCategories() függvénnyel az összes kategóriát törölhetjük a bejegyzésből.

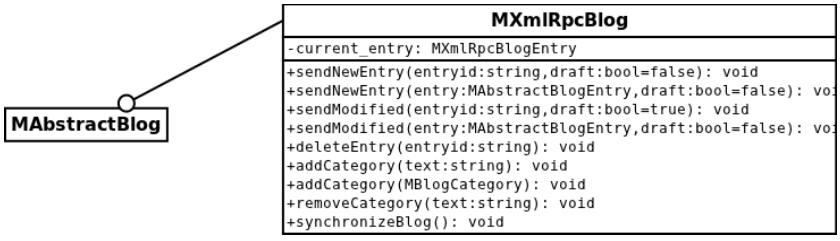
Az addTag(), removeTag() és a clearTags() függvények hatása hasonló a fenti függvényekéhez, de értelemszerűen ezek a bejegyzéshez tartozó címkéket kezelik.

A serializeEntry() virtuális metódus, működése a leszármazott osztályokban definiálandó. Feladata, hogy a bejegyzés adataiból egy olyan XML kimenetet állítson elő, amely a használt protokollnak megfelel. A blogmotorhoz intézett kérések során ezt az XML kimenetet kell elküldenünk.

6.2.5. MXmlRpcBlog

A korábban írtam, hogy a program első körben az XML-RPC alapú publikációs protokollt fogja támogatni. Az MXmlRpcBlog, és az MXmlRpcBlogEntry osztály feladata, hogy ezt a publikációs protokollt megvalósítsa.

Az MXmlRpcBlog osztályban tehát az XML-RPC protokoll kezelését fogom implementálni. Az MXmlRpcBlog osztály az MAbstractBlog osztály leszármazottja, mint ahogyan azt a 31. ábra is mutatja.



29. ábra. MXmlRpcBlog

A fentieknek megfelelően az osztály csak átdefiniált függvényeket és adattagokat tartalmaz. A current_entry adattag az éppen aktuális bejegyzés, ennek típusa MXmlRpcBlogEntry, melyről mindjárt szó lesz.

A sendNewEntry() és a sendModifiedEntry() metódusok bejegyzések beküldésére használhatóak. A beküldendő bejegyzés állapotától (flags adattag értéke) függ, hogy az adott pillanatban melyik függvény hajtódik végre.

A sendNewEntry() függvénynek két alakja van, a sendNewEntry(entryid,draft) függvénnyel közvetlenül az adatbázisból kezdeményezhetjük a bejegyzés beküldését, míg a sendNewEntry(entry,draft) függvénnyel egy, már előzőleg betöltött poszttal (például a current_entry-t) tehetjük meg ugyanezt. Mindkét esetben a draft paraméter szerepe ugyanaz: ha értéke true, a bejegyzést piszkozatként kell elküldeni, egyébként pedig normál bejegyzésként (tehát a bejegyzés rögtön meg is jelenik a blog főoldalán). A sendNewEntry() a metaWeblog.newPost, vagy a blogger.newPost távoli eljárást hívja meg.

A sendModifiedEntry() függvény működése hasonló a sendNewEntry()-éhez, a paraméterek jelentése is teljesen ugyanaz. Csak a függvény szerepe más, mivel ezzel a függvénnyel egy – már létező – bejegyzés módosítását tudjuk végrehajtatni a blogmotorral. Van egy másik lényeges különbség, amely azonban a függvény működésében jelentkezik majd, hogy egy beküldött bejegyzés esetén már ismerjük a postid-t, amivel a blogmotor a bejegyzést azonosítani tudja. A sendModifiedEntry() egyébként a metaWeblog.editPost, vagy a blogger.editPost távoli eljárást hívja meg.

A deleteEntry() függvénnyel egy publikált, és a blogmotor adatbázisában is szereplő bejegyzés törölhetünk. A deleteEntry egyetlen paramétere a bejegyzés BlogDB-beli azonosítója, ez azt jelzi, hogy a függvény közvetlenül az adatbázisból fog dolgozni. A blogból törölt bejegyzés a BlogDB adatbázisából nem fog törlődni, csak a flaget kell átállítani fles-Deleted-re, így a bejegyzés a program saját lomtárába kerül (mellesleg ugyanígy a blogmotor saját lomtárában is ott lesz, már ha a blogmotor rendelkezik ilyennel). A metódus a metaWeblog.deletePost eljárást fogja meghívni.

Az addCategory() függvénnyel egy új kategóriát hozhatunk létre a távoli blogban, de ez csak akkor történhet meg, ha a blogmotor támogatja a kategóriák kezelését. A metódus mindkét alakja a wp.newCategory távoli eljárást használja majd.

A synchronizeBlog() függvény működését az algoritmusok leírásánál fogom részletesen bemutatni.

6.2.6. MXmlRpcBlogEntry

Az MXmlRpcBlogEntry osztály az MAbstractBlogEntry leszármazottja. Az örökölteken kívül mindössze egyetlen saját tagfüggvénnyel rendelkezik, melynek neve serializeEntry().

Ez a függvény állítja elő a bejegyzés XML-RPC protokollnak megfelelő, XML formátumú leírását.

6.3. Az osztályban használt algoritmusok

6.3.1. Bejegyzéslista szűkítése

A osztály kapcsán a legfontosabb algoritmus a bejegyzések különböző feltételek szerinti szűrése. A következőkben ezt az algoritmust szeretném ismertetni.
A főablak „Bejegyzések” lapján lévő bejegyzéslistában időrendi sorrendben találhatók a bejegyzések, azonban alaphelyzetben más csoportosítási szempont nincs. Idővel, ha a blog már rengeteg posztot tartalmaz, ez a lista terjedelmes és ezáltal kicsit áttekinthetetlen lehet. A problémára megoldás lehet a bejegyzések listájának különböző szempontok szerinti szűkítése.
A legfontosabb szűrési szempontok:

- a bejegyzés tartalmában szereplő szövegrész,
- a bejegyzéshez tartozó kategóriák, illetve
- a bejegyzéshez tartozó címkék

Fontos megjegyezni, hogy azok a bejegyzések, amelyek egyetlen kategóriába vagy címkébe sem sorolhatók, mindenképpen megjelennek a szűkített listában is. Ez egy egyszerű megfontolás, könnyedén megváltoztatható, ha szükséges lenne.
Következzen maga az algoritmus...

```
// Bemenet
blog.entries : list<strEntryData> // a bejegyzések listája
temp.entries : list<strEntryData> // a feltételeknek megfelelő bejegyzések
szovegresz : string                       // a keresett szövegrész
kategoriak: list<MBlogCategory>   // a feltételként megadott kategóriák
cimkek: list<MBlogTag>            // a feltételként megadott címkék

;bejegyzések betöltése
// szövegrész szerinti szűrés
ha (kell szövegrész szerinti szűrés) {
        minden ed-re blog.entries-ben {   // ahol ed : strBlogEntryData
        ha (ed.content tartalmazza a szövegrészt) {
                ;ed hozzáfűzése temp.entries-hez
        }
        ;blog.entries törlése
        ;temp.entries mozgatása blog.entries-be
}
// kategóriák szerinti szűrés
ha (kell kategóriák szerinti szűrés) {
        minden ed-re blog.entries-ben {   // ahol ed : strBlogEntryData
                ha (ed kategóriának száma 0) {
                        ;ed hozzáfűzése temp.entries-be
                }
                egyébként {
                        minden bl-re entry.categories-ben {
                                        // ahol bl : MBlogCategory
                                minden cn-re kategoriak-ban {
```

```
                                        // ahol cn: MBlogCategory
                                        ha cn megegyezik bl-lel {
                                                ;talált=igen
                                        }
                                }
                        }
                        ha (talált==igen) {
                                ed hozzáfűzése temp.entries-hez
                        }
                }
        }
        ;blog.entries törlése
        ;temp.entries mozgatása blog.entries-be
}
// címkék szerinti szűrés
ha (kell címkék szerinti szűrés) {
        minden ed-re blog.entries-ben {   // ahol ed : strBlogEntryData
                ha (ed címkéinek száma 0) {
                        ;ed hozzáfűzése temp.entries-be
                }
                egyébként {
                        minden bl-re entry.tags-ben {       // ahol bl : MBlogTag
                                minden tn-re cimkek-ben {  // ahol tn: MBlogTag
                                        ha tn megegyezik bl-lel {
                                                ;talált=igen
                                        }
                                }
                        }
                        ha (talált==igen) {
                                ed hozzáfűzése temp.entries-hez
                        }
                }
        }
        ;blog.entries törlése
        ;temp.entries mozgatása blog.entries-be
}
// Kimenet
blog.entries, tartalma: azok a bejegyzések, amelyek megfeleltek a követelményeknek, vagy amelyeknek
egyetlen kategóriájuk, vagy címkéjük sincs
```

6.3.2. Szinkronizáció

A szinkronizáció során összehangoljuk a blogmotor adatbázisának tartalmát a program adatbázisával. Először a távoli blogmotorban létrehozott kategóriákat kell letölteni, és elmenteni a BlogDB-be. Ezután az adatbázisunkban létrehozásra kijelölt kategóriák beküldésével kell újra megpróbálkozni.

Ha ezzel megvagyunk, következhetnek a bejegyzések. Elsőként a blog által tárolt bejegyzéseket kell letöltenünk, és ellenőriznünk, hogy megvannak-e a blogadatbázisban. Ha nincsenek, értelemszerűen létre kell őket hozni. Mindeközben egy listában el kell tárolnunk a letöltött bejegyzések postid-jét.

Ezek után a blogból már törölt bejegyzéseket kell a lomtárba mozgatnunk. Ezt úgy tehetjük

meg, hogy minden egyes, BlogDB-ben tárolt, beküldöttként jelölt bejegyzés postid-jét le kell ellenőriznünk, hogy tartalmazza-e a letöltött bejegyzések postid-jeiből álló lista. Ha nem, a bejegyzést töröltként kell megjelölni. Ezután a blog adatbázisában beküldésre, módosításra, vagy törlésre kijelölt bejegyzésekre vonatkozó, korábban kiadott, de sikertelen kéréseket kell újra megismételnünk.

Alább következik az algoritmus, egy kicsit szemléletesebben...

```
// Bemenet
blog.entries – a blog adatbázisában eredetileg tárolt bejegyzések
blog.categories – a blog adatbázisában eredetileg tárolt kategóriák

// kategóriák letöltése és mentése
ha (a blogmotor támogatja a kategóriák letöltését) {
        ;mt.getCategoryList --> categories        // ahol categories:list<MBlogCategory>
        minden elem-re categories-ben {
                ;elem mentése adatbázisba
        }
}
;kategóriák újratöltése
// kategóriák létrehozása, törlése
ha (a blogmotor támogatja a kategóriák kezelését) {
        minden tmp-re blog.categories-ben {
                ha (tmp.flags==flcsMarkedForSubmit) {
                        // beküldésre kijelölt kategória
                        ;kategória beküldése
                } egyébként {
                        // törlésre kijelölt kategória
                        ;kategória törlése
                }
        }
        // létező kategóriák letöltése
        ;mt.getCategory --> categories2
                // ahol categories2:list<MBlogCategory>
        minden elem-re blog.categories-ben {
                ha (categories2 nem tartalmazza elem-et) {
                        ;elem kategória törlése
                }
        }
}
// bejegyzések letöltése
ha (a blogmotor támogatja a bejegyzések lekérését) {
        ;metaWeblog.getRecentPost --> tmp_entries
        // ahol tmp_entries : list<strBlogEntryData>
        minden entry-re tmp_entries-ben {
                ;entry postid-jének hozzáadása postids listához
                ;entry állapotának beállítása flesSubmitted-re
                minden ed-re blog.entries-ben {
                        ha (ed megegyezik entry-vel) {
                                ;ed adatainak frissítése
                                ;ed mentése az adatbázisba
                        }
                }
                ha (entry új bejegyzés) {
```

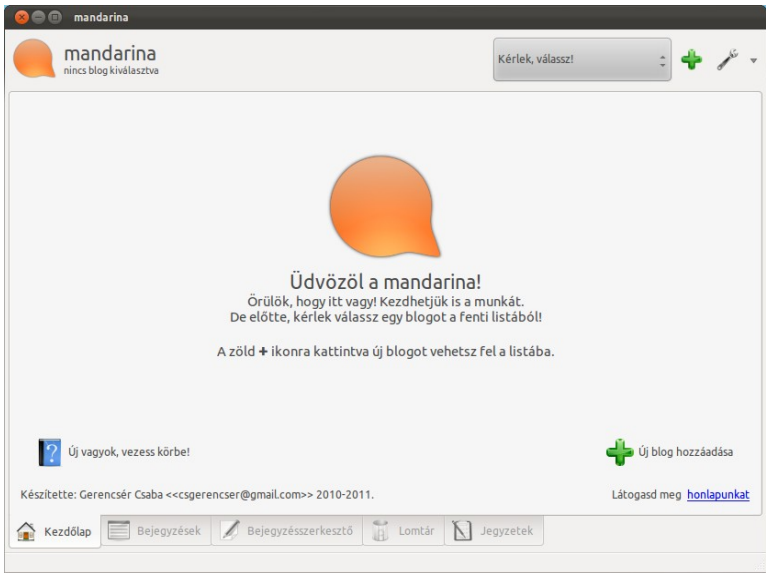
```
                        ;entry mentése új bejegyzésként
                }
        }
}
;bejegyzések újratöltése
// blogban már nem szereplő bejegyzések lomtárba helyezése
minden ed-re blog.entries-ben {
        ha (ed.postid nem szerepel postids-ben) {
                ed állapotának beállítása flesDeleted-re
        }
        ha (ed.postid szerepel postids-ben, de ed.flags értéke flesDeleted)        {
                ;ed állapotának beállítása flesSubmitted-re
        }
}
;bejegyzések újratöltése
// korábban félbeszakadt bejegyzésműveletek ismétlése
minden ed-re blog.entries-ben {
        ha (ed.flags értéke flesMarkedForSubmit)
                ;bejegyzés beküldése              // sendNewEntry()
        ha (ed.flags értéke flesMarkedForModify)
                ;bejegyzés módosítása             // sendModifiedEntry()
        ha (ed.flags értéke flesMarkedForDelete)
                ;bejegyzés törlése                // deleteEntry()
        ha (ed.flags értéke flesMarkedForRecover)
                ;bejegyzés visszaállítása         // sendModifiedEntry() !!!
}
// Kimenet
blog.entries, és blog.categories, az új, illetve módosított bejegyzésekkel, kategóriákkal
```

7. A kész program ismertetése

A következő fejezetben a kész programot szeretném bemutatni, működés közben. A tudnivalókat igyekeztem úgy megfogalmazni, hogy a leírtakból bárki, pillanatok alatt megtanulja a program használatát.

7.1. A kezdőképernyő

A 32. ábrán a program indítóképernyője látható.



30. ábra. A kezdőképernyő

Az ablak címsora alatt található a blog fejléce. Mivel nincs aktív blog, a blogikon helyén a program ikonja, a blog becenevének helyén a program neve szerepel.

Mellette a bloglistát láthatjuk. A program indításakor ez természetesen üres. A listába új blogot felvenni a lista melletti zöld ikonra, vagy a kezdőképernyő alsó részén lévő, „Új blog hozzáadása” nyomógombra kattintva lehet. Ennek hatására az „Új blog hozzáadása” nevű párbeszédablak töltődik be, melyet később ismertetek.

A mellette lévő ikon a program főmenüje. Az ikonra kattintva a „Beállítások” párbeszédpanel indul. A lefelé mutató háromszögre kattintva megjelenik a főmenü. Sajnos erről nem tudtam képernyőképet készíteni, így csak felsorolnám a menüpontokat:

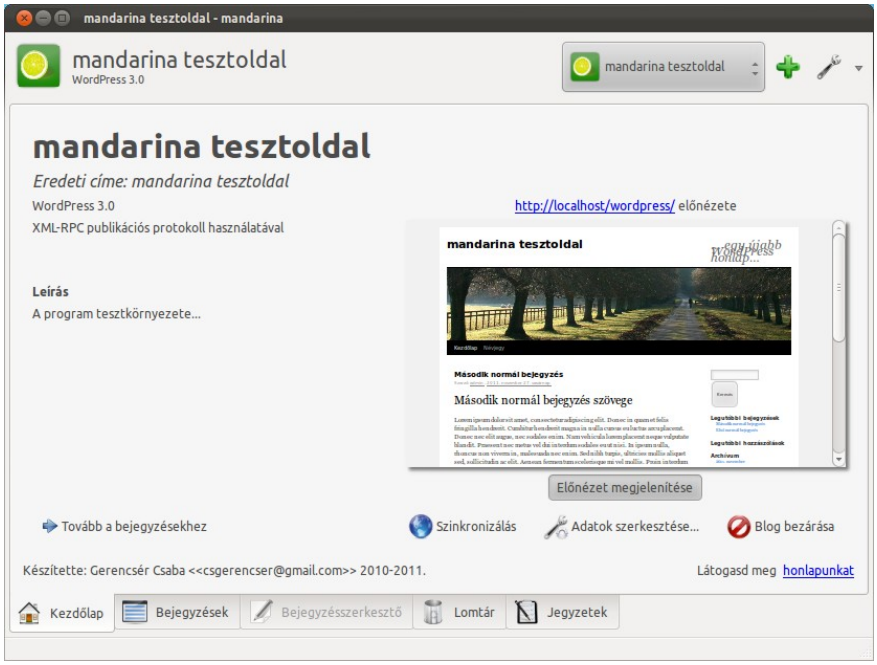
- „Kapcsolat nélküli munka” - ha a menüpont aktív, a program nem férhet hozzá a számítógép internetkapcsolatához
- „Beállítások” - hatása ugyanaz, mintha a kulcs ikonra kattintanánk, megjelenik a „Beállítások” párbeszédablak

– „Súgó” - részletes segítség a program használatához
– „A mandarina névjegye” - információk a programról
– „A Qt névjegye” - információk arról a Qt verzióról, amivel a program adott verziója fordítódott
– „Kilépés” - az aktív blog bezárása, és kilépés a programból

Az ablak középső részén a program ikonjának nagyméretű változata látható, egy rövid, üdvözlő szöveggel. Az alatta lévő két nyomógomb közül az egyiket már ismerjük. A másik, „Új vagyok, vezess körbe!” feliratú gombra kattintva egy rövid összefoglaló leírás jelenik meg a program használatáról.

Ezek alatt található az oldalsáv. Indításkor a „Kezdőlap” nevű fülön kívül az oldalsáv összes eleme inaktív. Aktívvá csak akkor válhatnak, ha a bloglistából a felhasználó kiválaszt egy blogot.

Ha ez megtörtént, a 33. ábrához hasonló látvány tárul elénk.



31. ábra. A kezdőképernyő egy blog kiválasztása után

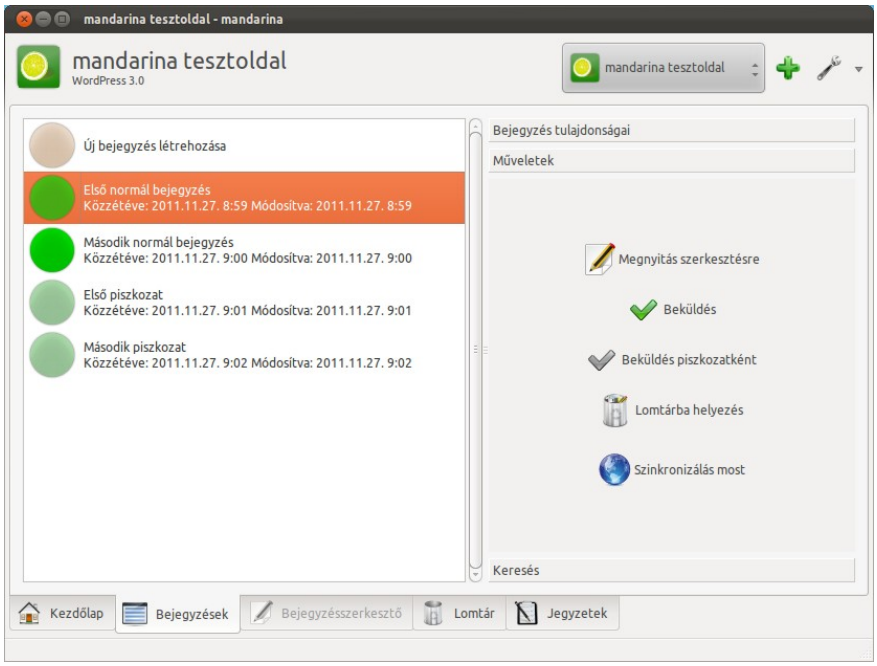
A középső mező három részre oszlik. A bal oldali területen a blog adatai láthatóak, úgy, ahogyan azokat a program a regisztrációkor kiderítette, illetve ahogy a felhasználó megadta. A jobb oldalon a blog előnézeti képe jelenik meg, egy miniatűr böngészőben. Az előnézet megjelenítését a böngésző alatti nyomógombbal kapcsolhatjuk ki vagy be.

Az alsó részen egy gombsor látható. Az első, „Tovább a bejegyzésekhez” feliratú nyomó-

gombra való kattintással a „Bejegyzések” lapra ugrunk. A „Szinkronizálás” gombbal elindul a program adatbázisának és a távoli blog tartalmának összehangolása. Ezt egy blog felvétele után érdemes megtenni. Az „Adatok szerkesztése” gombbal megjelenik a „Blog tulajdonságainak szerkesztése” című párbeszédablak, ahol a regisztrációkor megadott bizonyos adatokon tudunk változtatni. A „Blog bezárása” gombra kattintva a program bezárja az éppen aktív blogot, és visszatér a kiindulási állapotba.

7.2. A bejegyzéslista

A bejegyzések listája látható a 34. ábrán.



32. ábra. A bejegyzéslista

A felhasználói felület két részre oszlik. Bal oldalon látható a bejegyzéslista. Az egyes bejegyzések pillanatnyi állapotát a mellettük látható kör alakú ikonok segítségével lehet nyomon követni. Az egyes ikonok és jelentésük az alábbi táblázatban látható. A törölt, a törlésre kijelölt és a visszaállításra kijelölt bejegyzések ikonjai nem ebben a listában, hanem a „Lomtár” fül hasonló listáján jelennek meg, de a teljesség igénye miatt ebben a táblázatban szerepelnek.

Új bejegyzés

Helyi piszkozat

 Beküldésre kijelölt normál bejegyzés

 Módosításra kijelölt normál bejegyzés

 Beküldött normál bejegyzés

 Törlésre kijelölt bejegyzés

 Visszaállításra kijelölt normál bejegyzés

 Törölt bejegyzés

 Beküldésre kijelölt piszkozat

 Módosításra kijelölt piszkozat

 Beküldött piszkozat

 Visszaállításra kijelölt piszkozat

A lista első elemének neve „Új bejegyzés létrehozása”, ha erre duplán kattintunk, a „Bejegyzésszerkesztő” fül aktívvá válik, és egy új bejegyzés írásához kezdhetünk. A többi elemre való dupla kattintással az adott bejegyzés fog betöltődni a bejegyzésszerkesztőbe.

33. ábra

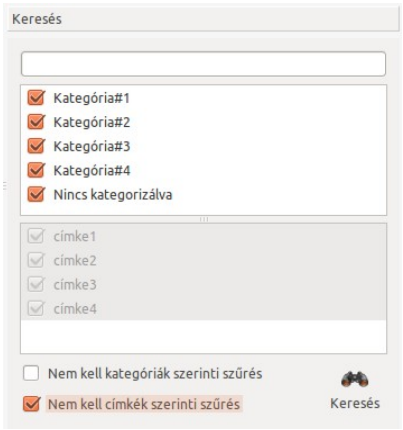
Jobb oldalon, a műveletsávon a bejegyzésekhez kapcsolódó műveleteket találhatjuk. A gombokból álló lista tartalma attól függően változik, hogy milyen státusszal rendelkezik az éppen kiválasztott bejegyzés. Kivételt ez alól a „Megnyitás szerkesztésre” és a „Szinkronizálás” gomb képez. Előbbivel a kijelölt bejegyzést nyithatjuk meg, utóbbival pedig a kezdőképernyőn található gombhoz hasonlóan a szinkronizálást indíthatjuk el. A program folyamatosan tájékoztat bennünket a kiválasztott esemény állapotáról, egyrészt az állapotsoron megjelenő üzenetekkel, másrészt (kritikus hiba esetén) figyelmeztető ablakokkal, harmadrészt pedig a blogikon megváltoztatásával. Folyamatban lévő műveletek esetén egy forgó fénycsóvával, sikeresség esetén egy zöld pipával, figyelmeztetés esetén egy sárga felkiáltójellel, hiba esetén pedig egy piros kereszttel. A 35. ábrán a blog ikonjának sikeres művelet utáni viselkedését láthatjuk.

Ha egy bejegyzéssel kapcsolatban több információra vágyunk, mint ami a listában megjelenik, a műveletsáv „Bejegyzés tulajdonságai” feliratú paneljét kell kiválasztanunk. Ilyenkor az éppen kijelölt bejegyzésről láthatunk egy kis összefoglalót, a 36. ábrán látható módon.

34. ábra. Bejegyzés tulajdonságai

Lehetőség nyílik arra is, hogy a bejegyzések között keressünk, szövegrészlet, címrészlet, kategóriák, illetve címkék szerint. Ehhez a műveletsáv „Keresés” nevű paneljét használhatjuk. A panelt a 6. ábrán szemügyre vehetjük.

A felső szövegmezőbe írhatjuk a keresett kifejezést. A kereső pontos kifejezésekre keres, viszont a kisbetűk és a nagybetűk azonosnak számítanak. A szövegmező alatti listákban a blog kategóriái (felső lista), valamint a címkéi (alsó lista) vannak felsorolva. A kívánt kategóriák kijelölése úgy történik, hogy aktiváljuk a nevük előtti jelölőnégyzetet. Ha végeztünk a kritériumok összeállításával, a lista szűkítését a „Keresés” gombra kattintva indíthatjuk.



35. ábra. Bejegyzések keresése

Lehetőség van arra is, hogy bizonyos kritériumokat kizárjunk a keresésből. A szövegrész szerinti keresés kizárásához elég, ha üresen hagyjuk a beviteli mezőt. A kategóriák vagy címkék szerinti keresés kihagyásához a nyomógomb melletti jelölőnégyzeteket kell deaktiválni. Ilyenkor a megfelelő kategória- vagy címkelista a képen látható módon kiszürkül.

A keresés befejezése után a bejegyzéslista csak azokat a bejegyzéseket tartalmazza, amelyekben szerepel a megadott kifejezés ÉS tartalmazzák a kijelölt kategóriákat és címkéket,

valamint azokat a bejegyzéseket, amelyek egyetlen kategóriához vagy címkéhez sem tartoznak (árva bejegyzések).

7.3. A lomtár

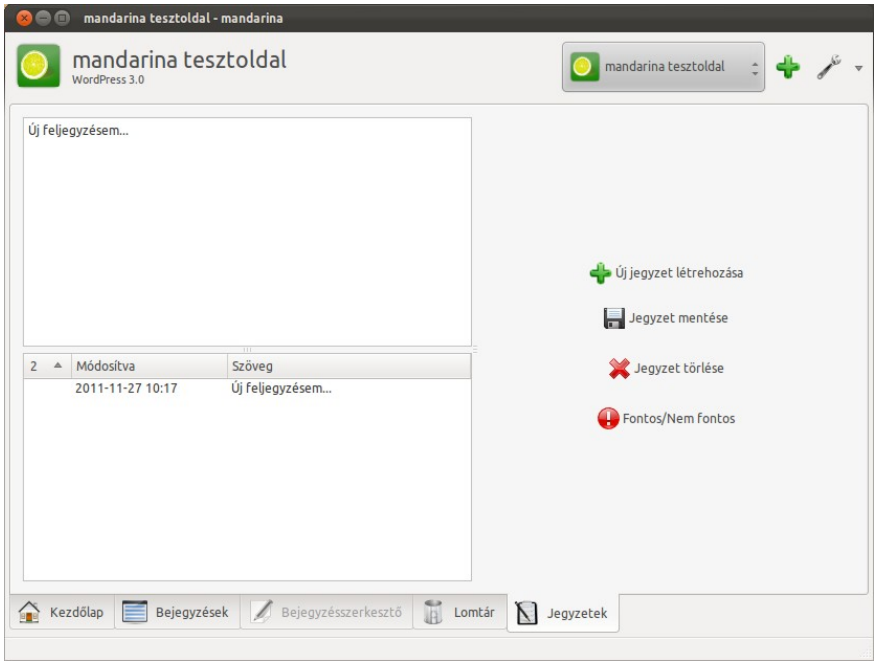
Azok a bejegyzések, amelyeket helyi piszkozatként, vagy beküldött bejegyzésként a bejegyzéslista „Lomtárba helyezés” gombjával töröltünk, a program lomtárába kerülnek. A lomtárt az oldalsáv „Lomtár” fülét választva érhetjük el. A lomtár külalakja megegyezik a bejegyzéslistáéval, a különbség mindössze annyi, hogy a műveletsávon kevesebb opció érhető el, illetve a bejegyzések között nem lehet keresni. A műveletsávon csak három opció található:

- „Teljes visszaállítás”: az előzőleg törölt bejegyzést újra elküldi a blogmotornak, így a bejegyzés újra megjelenhet a blog főoldalán, vagy a bejegyzések között.
- „Visszaállítás helyi piszkozatként”: a törölt bejegyzés helyi piszkozat lesz, vagyis egy szerkeszthető, és a későbbiekben újra elküldhető bejegyzés
- „Végleges törlés”: a bejegyzés véglegesen törlődik a program adatbázisából

7.4. A jegyzettömb

A mandarina saját, beépített jegyzettömbbel is rendelkezik. Ez a funkció leginkább emlékeztetők írására használható. A program nemcsak a jegyzet szövegét, de az utolsó módosítás dátumát is elmenti. Emellett lehetséges az is, hogy bizonyos feljegyzéseket fontosként megjelöljünk.

A jegyzettömb a 38. ábrán látható.



36. ábra. A jegyzettömb

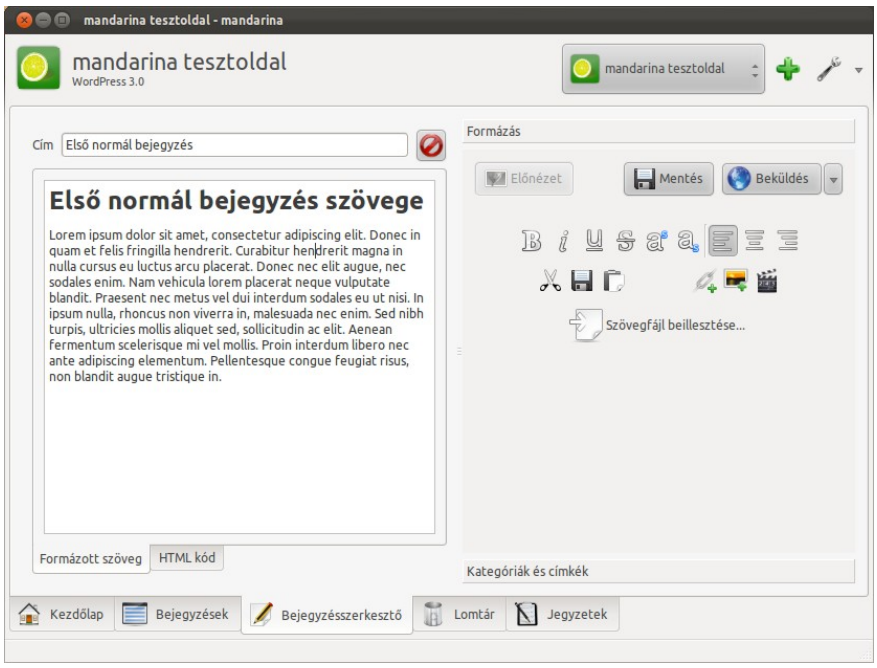
Az ablak bal oldalán egy szövegszerkesztő és egy táblázat látható. A szövegszerkesztő mezőbe az aktuálisan szerkesztett jegyzet szövege kerül. Az alatta lévő táblázatban a bloghoz tartozó jegyzetek listája jelenik meg. A táblázat fejlécére kattintva a jegyzetek fontosság, módosítási dátum, valamint tartalom szerint sorrendbe rendezhetők. Ha egy jegyzet fontosként lett megjelölve, az első oszlopban egy piros felkiáltójel jelenik meg.

A műveletsávon mindig négy nyomógomb van. Az elsővel új jegyzetet készíthetünk. A második gombbal az éppen szerkesztett jegyzet módosításait tudjuk elmenteni. A harmadik nyomógombbal a kijelölt jegyzetet véglegesen törölhetjük. Az utolsó gomb megnyomásával megadhatjuk, hogy a kijelölt jegyzet fontos-e avagy sem.

7.5. A bejegyzésszerkesztő

Ha a bejegyzéslistában egy bejegyzésre duplán kattintunk, vagy megnyomjuk a „Megnyitás szerkesztésre” nyomógombot, a bejegyzés betöltődik a memóriába, mi pedig a „Bejegyzésszerkesztő” nevű lapra kerülünk.

A bejegyzésszerkesztő a következőképpen néz ki.



37. ábra. A bejegyzésszerkesztő

Hasonlóképpen az eddigiekhez, a bejegyzésszerkesztő területe két részre oszlik. A bal oldalon található a bejegyzés tartalma. A „Cím” feliratú beviteli mezőben a bejegyzés címe adható meg. Az alatta lévő szövegszerkesztő két fülből áll. Az első fülön egy Word-jellegű szövegszerkesztő található, amely képes a speciális formázások megjelenítésére is. A második egy HTML szerkesztő, a bejegyzést HTML kód formájában írhatjuk be.

A jobb oldalon a műveletsávon a bejegyzések formázásához használható eszköztár, valamint a bejegyzéshez tartozó kategóriák és címkék megadására szolgáló panelek találhatóak.

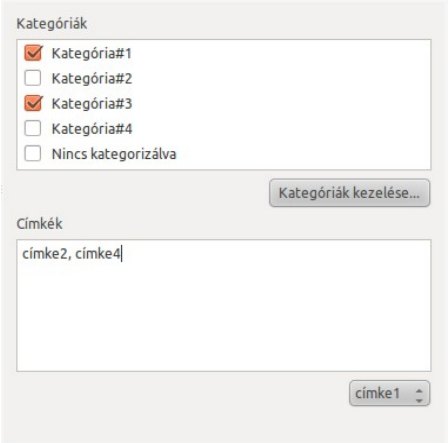
A formázó eszköztár első sorában a karakterekre és bekezdésekre értelmezhető lehetőségek szerepelnek, vagyis az alapvető betűstílusok (félkövér, dőlt, aláhúzott, áthúzott, felső és alsó index) és a bekezdés igazítása (balra, jobbra és középre igazítása). A második sor első részében a szokásos vágólapműveletek, a második részében pedig a különböző hivatkozások beillesztését segítő párbeszédablakok „előcsalogatására” szolgáló gombok találhatóak. A harmadik sorban szereplő egyetlen gombbal egy kész szövegfájl tartalmát tudjuk a bejegyzés szövegébe illeszteni.

A „Mentés” gombbal a bejegyzés tartalmának változásait tudjuk elmenteni, a „Küldés” gombbal pedig közvetlenül elküldhetjük a bejegyzést a blogmotornak. A piros, áthúzott ikonra kattintva bezárhatjuk a bejegyzést, és visszatérhetünk a bejegyzéslistához.

A műveletsáv másik panelén, melynek „Kategóriák és címkék” a neve, a bejegyzéshez tar-

tozó kategóriákat és címkéket tudjuk kijelölni. A panel a 9. ábrán látható.

A kategórialistában a program adatbázisában szereplő összes kategória helyet kap. Ahhoz, hogy a szerkesztett bejegyzést besoroljuk egy kategóriába, a felső listában a kívánt kategória neve előtti jelölőnégyzetet kell aktiválnunk. Új kategóriák létrehozása vagy törlése csak a „Kategóriaszerkesztő” nevű párbeszédablakban lehetséges, ezt a „Kategóriák kezelése” feliratú gombbal jeleníthetjük meg.

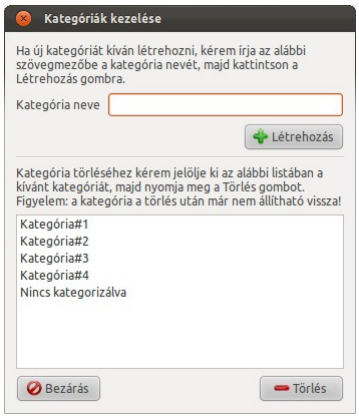


38. ábra. Kategóriák és címkék megadása

A címkelista kezelése jóval egyszerűbb. Ha egy címkét akarunk a bejegyzéshez kapcsolni, elég beírni a nevét. Több címkét is megadhatunk, az egyes címkéket vesszővel vagy Enter-rel tudjuk elválasztani. Ha a listát már létező címkével akarjuk bővíteni, a lista alatti legördülő menüből kiválaszthatunk egyet. Ha olyan címkét akarunk a bejegyzéshez hozzáadni, amely már szerepel a felsorolásban, az állapotsorban figyelmeztetést kapunk.

7.6. A kategóriaszerkesztő

Ha az előbb említett nyomógombra kattintunk, a következő panel jelenik meg.



39. ábra. A kategóriaszerkesztő

Kategória létrehozásához először be kell írni a kívánt nevet, majd a „Létrehozás” gombra kell kattintani. Amennyiben a kategória létrehozásának pillanatában a program hozzáfér az internethez, a kategória a távoli blogban is létrejön. Ha nincs internetkapcsolat, a kategóriát a program létrehozandóként jelöli meg. A kategóriát ténylegesen létrehozó hálózati kérést a bejegyzés elküldésekor, vagy a legközelebbi szinkronizáció idején megismétli.

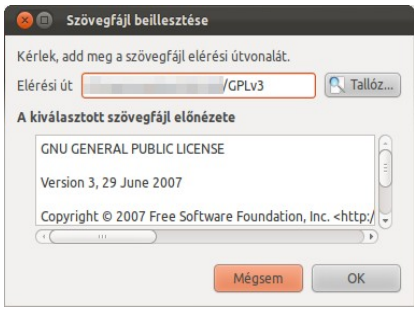
Kategória törléséhez a lenti listából ki kell választanunk a megfelelő elemet, ezután a „Törlés” gombot kell megnyomnunk. A létrehozáshoz hasonlóan a törlés is aktív internetkapcsolatot igényel.

A párbeszédpanel a „Bezárás” gombra kattintással tüntethető el.

7.7. Fájlok beillesztése

A formázó eszköztár „Szövegfájl beillesztése” feliratú gombjára való kattintással a következő párbeszédablakot tudjuk elérni.

A „Szövegfájl beillesztése” nevű panel segítségével bármilyen nyers szöveges állomány tartalmát illeszthetjük be az aktuális kurzorpozícióba.



40. ábra. Szövegfájl beillesztése

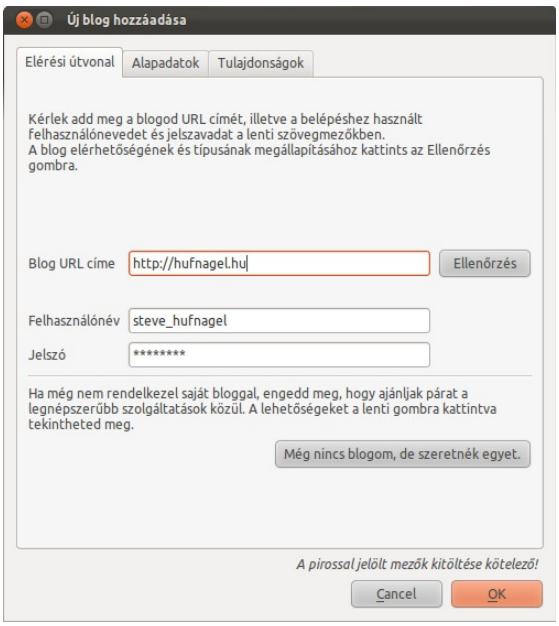
A szövegfájl kiválasztásánál vagy magunk írjuk be az állomány elérési útvonalát, vagy a „Tallóz” nyomógombra való kattintás után megjelenő ablakban kikeressük a fájl pontos helyét.

Annak érdekében, hogy biztosan a megfelelő állományt válasszuk ki, egy szövegdobozban az állomány tartalmának előnézete is megjelenik.

7.8. Új blog hozzáadása

Ahhoz, hogy a programot használni tudjuk, rendelkeznünk kell egy olyan bloggal, amellyel szót tud érteni a program, vagyis egy olyan bloggal, amely támogatja az XML-RPC alapú publikációs protokollt. Ha már van ilyenünk, akkor regisztrálnunk kell a mandarinában is. A regisztrációt a már említett módon, a blogfejlécben található zöld plusz jelre, vagy a kezdőoldalon található nyomógombra kattintva tudjuk elindítani.

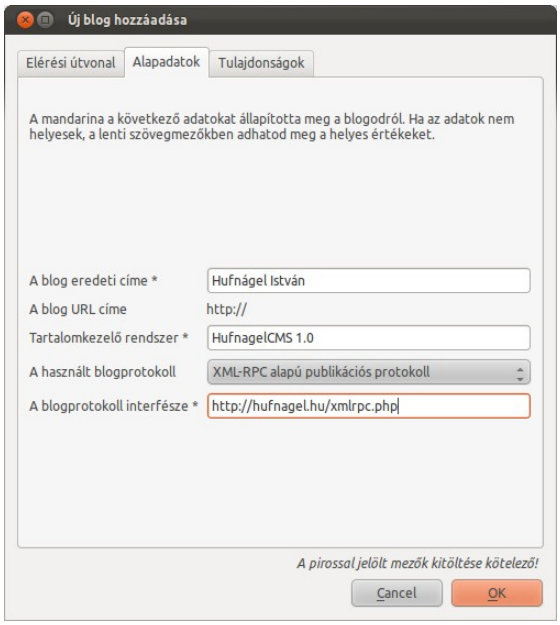
Ha ezt megtesszük, a 43. ábrán látható párbeszédablak vár ránk.



41. ábra. Új blog hozzáadása

A program lehetőséget biztosít arra is, hogy keressünk magunknak egy megfelelő blogkiszolgálót. Az ajánlatokat a „Még nincs blogom...” gombra kattintva tekinthetjük meg.

A panel első oldalán meg kell adnunk a blog eléréséhez használható legfontosabb adatokat, vagyis a blog URL címét, valamint azt a felhasználónevet és jelszót, amivel a blog adminisztrációs felületére be szoktunk lépni. Ha ezzel megvagyunk, nyomjuk meg az „Ellenőrzés” feliratú gombot. Ennek hatására a program kapcsolatba lép a bloggal, és a többi fontos adatot megpróbálja magától kitalálni, és a hiányzó mezőket ennek megfelelően kitölteni. Ha sikeres volt ez a művelet, lépjünk a második lapra, hogy az adatokat ellenőrizni tudjuk. Ha nem sikerült, akkor is érdemes, hogy a hiányzó adatokat megadhassuk.

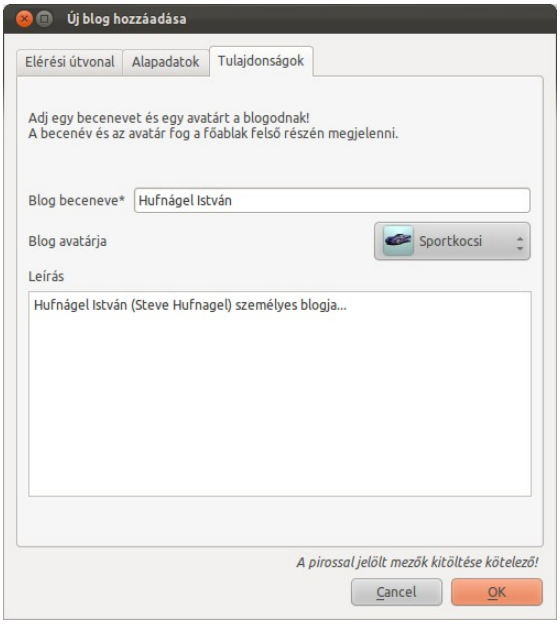


42. ábra. Új blog hozzáadása - 2. lap

A második oldalon az automatikus felismerés során kiderített adatok találhatóak:

- a blog eredeti címe, amely a webböngésző címsorában is megjelenne
- tartalomkezelő rendszer: a blogmotor neve, esetleg verziószáma
- a használt blogprotokoll: a blog eléréséhez használt publikációs protokoll, ez jelenleg csak az XML-RPC lehet
- a blogprotokoll interfésze: az a speciális URL cím, melyre a program a működése során kéréseket fog küldeni, illetve innen válaszokat fogadni

Ha úgy ítéljük meg, hogy az adatok stimmelnek, nyugodtan ugorjunk a következő lapra, ahol a blog programon belüli azonosításához szükséges adatokat adhatjuk meg.



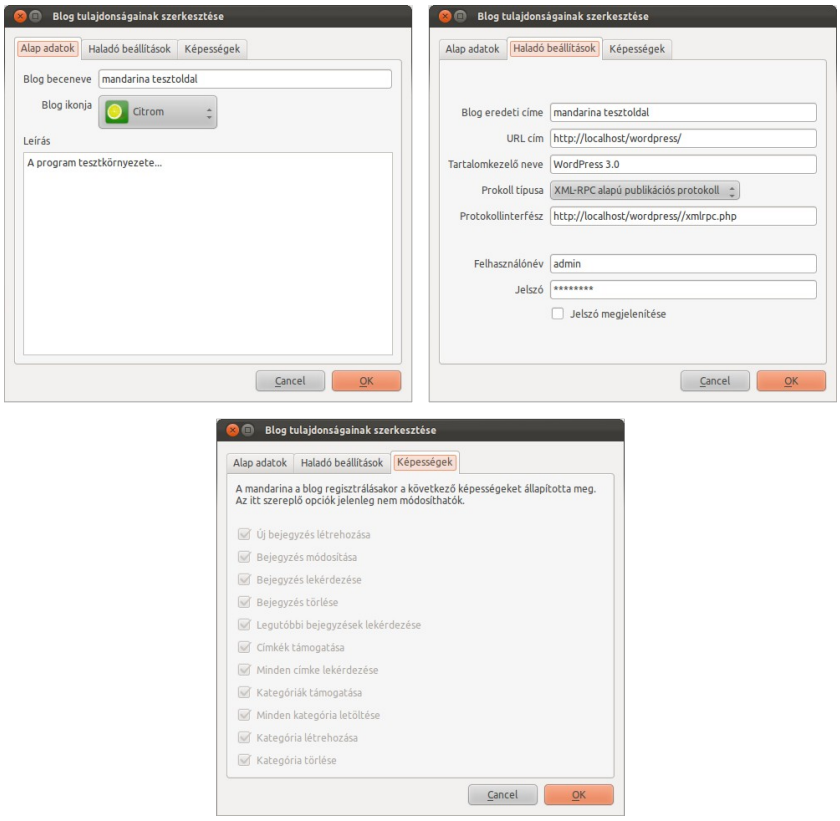
43. ábra. Új blog hozzáadása - 3. lap

A blog beceneve az a rövid cím, amely majd a program fejlécében is megjelenik, a blogikon mellett. Ez alaphelyzetben megegyezik a blog eredeti címével, de természetesen itt bármit megadhatunk, ami csak jólesik.

A blog ikonját (avatárját) az ikonokat tartalmazó legördülő menüből választhatjuk ki. A kész ikonokon kívül saját képfájlt is megadhatunk, a legördülő menü utolsó, „Egyéni...” lehetőségét választva.

Az alsó szövegdobozba egy hosszabb leírást írhatunk a blogról. Ez a leírás a blog kiválasztását követően a kezdőképernyőn jelenik meg.

7.9. Blog adatainak testreszabása



44. ábra. Blog tulajdonságok

Az „Új blog hozzáadása” panelen megadott adatokat utólag is módosíthatjuk. Ehhez a „Blog tulajdonságai módosítása” párbeszédablakot kell előhívnunk. Ez az ablak látható a 46. ábrán. A panel első oldalán a blog alap adatait változtathatjuk meg, vagyis a becenevet, a bloghoz tartozó ikont, és a leírást.

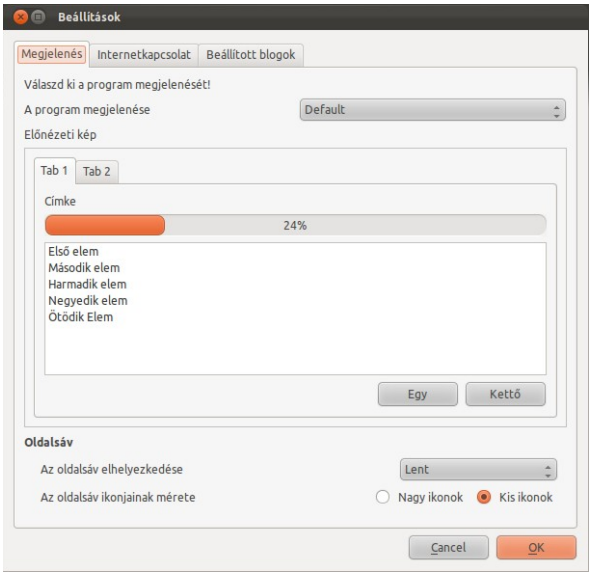
A második oldalon a bloggal történő kommunikációhoz szükséges adatok láthatóak. Itt lehetséges a blog URL címét, a használt tartalomkezelő nevét, a blogprotokoll típusát és a blogmotor eléréséhez használt interfész címét megváltoztatni. Továbbá, ha időközben megváltozik a felhasználónevünk és a jelszavunk, akkor az új adatainkat itt módosíthatjuk.

A harmadik oldalon a blogmotor képességeit láthatjuk. Ez azt jelenti, hogy a blogunk a szabványosnak tekinthető blogprotokoll mely szolgáltatásait ismeri, és melyeket nem. Ezeket az értékeket nem módosíthatjuk.

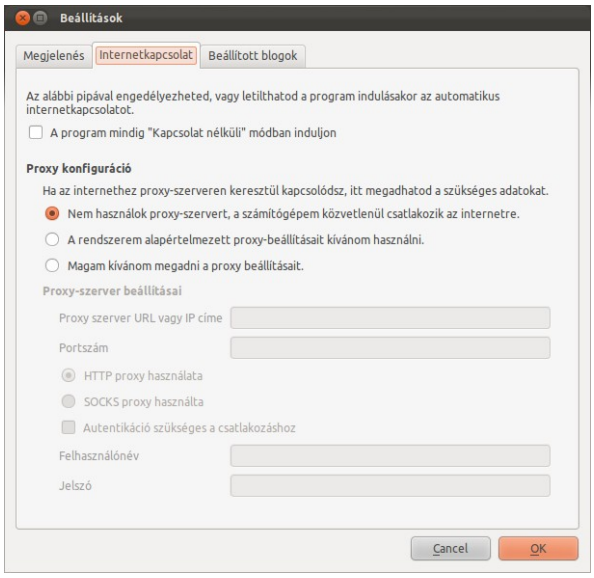
7.10. Beállítások

A programhoz tartozó beállítási lehetőségeket a „Beállítások” párbeszédpanel tartalmazza. Akkor jelenik meg, ha a blogfejlécben a menügombra kattintunk, vagy a gombhoz tartozó menüben a „Beállítások” menüpontot választjuk.

A párbeszédpanel első lapja a program megjelenésével kapcsolatos beállításoknak ad helyet. A „Megjelenés” nevű lap a 47. ábrán látható.



45. ábra. Beállítások - Megjelenés



46. ábra. Beállítások - Internetkapcsolat

A panel felső részén a program által használt témát állíthatjuk be. A kívánt témát a legördülő listából választhatjuk ki. A kiválasztott téma előnézeti képét a lista alatti területen tekinthetjük meg.

A panel alsó részén, az „Oldalsáv” nevű csoportban az oldalsávra vonatkozó beállítások vannak. Beállítható az oldalsáv elhelyezkedése (fent, lent, balra, jobbra), illetve az oldalsávon látható ikonok mérete.

A második lapon az internetkapcsolattal kapcsolatos beállítások jelennek meg. Ezeket a 48. ábrán láthatjuk.

A legfelső jelölőnégyzettel az automatikus kapcsolat nélküli üzemmódot állíthatjuk be. Ekkor a program, az indulásától kezdve kapcsolat nélküli módban dolgozik, a blogmotorhoz szóló kérések előtt külön kell engedélyezni a hálózathoz való hozzáférést.

A „Proxy konfiguráció” csoportban a proxy szerverre vonatkozó adatokat adhatjuk meg. Itt három lehetőség közül választhatunk. ha nem használunk proxy-t, az első rádiógombot kell

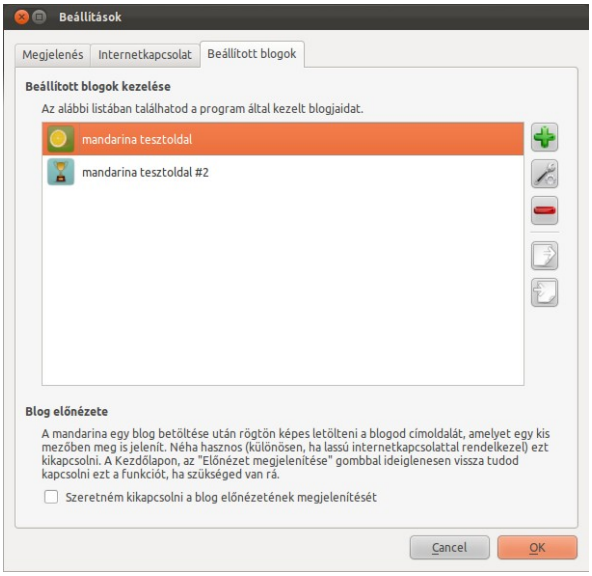
választanunk. Ha használunk proxy-t, de nem tudjuk fejből megadni az adatokat, (vagy kényelmesek vagyunk, és ezért nem is akarjuk megadni), a második rádiógomb kijelölése után a program az általunk használt operációs rendszer proxy-beállításait fogja használni.

Ha saját magunk szeretnénk az adatokat megadni, a harmadik rádiógombot kell választanunk. Ezután már meg tudjuk adni a következő adatokat: a proxy szerver címe, a csatlakozáshoz használt port száma, a proxy típusa (HTTP, vagy SOCKS5). Ha a proxy használatához autentikáció is szükséges, a megfelelő jelölőnégyzet bekapcsolása után megadhatjuk a felhasználónevet és a jelszót is.

A párbeszédablak harmadik lapján a programba regisztrált blogokkal kapcsolatos beállítások találhatóak. A lap két részből tevődik össze, mint ahogyan az a 49. ábrán is látható.

A felső csoportban a programba eddig regisztrált blogokkal végezhetünk különböző műveleteket.

A zöld + ikonra kattintva egy új blogot regisztrálhatunk. A csavarkulcs ikont választva a kijelölt blog adatait szerkeszthetjük a korábban már megismert párbeszédpanelen. A piros – ikonra kattintva a kijelölt blogot véglegesen törölhetjük a programból.



47. ábra. Beállítások - Blogok

Az alattuk lévő két nyomógombbal a blogokhoz tartozó állományokat lehet exportálni, illetve importálni. Hasznos funkció ez, ha a blogokhoz tartozó adatbázisfájlokról szeretnénk biztonsági másolatot készíteni, illetve egy ilyen, korábban készített másolatot szeretnénk visszaállítani.

8. Továbbfejlesztési lehetőségek

A mandarina egy tipikus programnak tekinthető, abból a szempontból legalábbis biztosan, hogy sosem készül el igazából. Annak ellenére, hogy (az előkészületekkel) együtt valamivel több, mint egy évig dolgoztam rajta, még mindig vannak dolgok, amelyekkel elégedetlen vagyok, és vannak bizony olyan dolgok is, amelyeket idő hiányában nem tudtam kivitelezni.

Ebben a fejezetben szeretném összegezni, hogy a program a jövőben merre fejlődhetne tovább.

8.1. További protokollok támogatása

A program jelenleg csak az XML-RPC alapú publikációs protokollt támogatja. Mint ahogy írtam, azért esett a választásom erre a protokollra, mert ez tűnt a legegyszerűbben kivitelezhetőnek. Más protokollok, mint amilyen az Atom Publishing Protocol, implementálása nagyobb falat lett volna, bár az is lehet, hogy célravezetőbb.

Az XML-RPC protokoll ugyanis rendelkezik néhány hiányossággal, amit nem lehet elhallgatni:

- Az XML-RPC a fejlesztők minden igyezete ellenére nem teljesen szabványos, ezért kellett egy új blog felvétele előtt mindig ellenőrizni, hogy a blogmotor a protokoll mely részét ismeri, melyiket nem.
- A használt kommunikációs csatorna nem biztonságos. A kérések általában a HTTP protokollon keresztül „utaznak”, de ez egy titkosítás nélküli protokoll. A blogmotorok is általában a titkosítás nélküli HTTP-t részesítik előnyben. Ezzel csak az a probléma, hogy egy kérésben mindenféle titkosítás nélkül megtalálható a felhasználó jelszava.

A blogkezelő objektumokat (MAbstractBlog, MAbstractBlogEntry) igyekeztem úgy megtervezni, hogy a későbbiekben az MXmlRpcBlog osztály mintájára újabb osztályokat lehessen belőlük származtatni, amelyekkel azután meg lehetne valósítani az AtomPub, vagy a Google Blogger API támogatását.

8.2. Többszálúsítás

A program az egyszerűség érdekében elméletileg egyetlen szálon fut. Azért csak elméletileg, mert a gyakorlatban (a Qt QNetwork moduljának köszönhetően) a hálózati kérések külön szálon működnek, aszinkron módon. Az adatbázisműveletek viszont nem. Emiatt az időigényesebb műveletek, mint amilyen a bejegyzések mentése, vagy elküldése, egy időre blokkolják a program futását, ez idő alatt a felhasználói felület nem reagál semmilyen eseményre. A szinkronizáció főleg nagyon csúnyán néz ki, hiszen felváltva, hol megakad, hol tovább folytatódik a folyamat, ahogy a hálózati kérések és az adatbázisműveletek egymást váltják.

A problémát igyekeztem megoldani, de végül visszakoztam, mert sok nagyobb mértékű

változtatást kellett volna elvégezni, ehhez viszont sok idő kellett volna.

8.3. Több bejegyzés kezelése

A mandarina jelenleg egy időben csak egy bejegyzés kezelésére képes. Elképzelhető, hogy néhány felhasználó ezt kevésnek találja; hiszen már megszokta más programok, például a szövegszerkesztő alkalmazások esetén, hogy egyszerre több dokumentumot is szerkeszthet, márpedig a mandarina lényegében egy speciális szövegszerkesztőnek is tekinthető.

A felhasználói felületen gyakorlatilag semmit sem kell változtatni. Az oldalsávon több bejegyzésszerkesztő lap is megjelenhet. Programozás szintjén ehhez a bejegyzésszerkesztő külön osztályba helyezése szükséges.

8.4. Médiakezelés

A modern tartalomkezelők, mint amilyen a WordPress, lehetőséget biztosítanak arra, hogy a felhasználó egy webes felületen fájlokat töltsön fel a blog tárhelyére, majd pedig arra, hogy ezeket a fájlokat a készülő bejegyzés szövegébe illessze. Szerencsénkre az ilyen fájlokat XML-RPC protokollon keresztül is lehet kezelni. Nagyon kényelmes lenne a felhasználó szempontjából, ha közvetlenül a programból tudna képeket, videókat csatolni a készülő bejegyzéséhez.

A WordPress a wp.uploadFile, wp.getMediaLibrary és a wp.getMediaItem metódust bocsátja a programozó rendelkezésére.

8.5. Képek és videók beágyazása

A képek és videók linkjeinek beágyazására létrehozott felhasználói felület jelenleg igen egyszerű, csak egy szövegdobozt és egy nyomógombot tartalmaz. Ez így, jelenlegi állapotában majdnem felesleges, hiszen egy HTML kódot a HTML szerkesztőbe is be lehet illeszteni... Kell valami szükséges plusz, ami miatt érdemes lehet ezt a funkciót használni.

A probléma megoldása a következő lehetne: a párbeszédpanelt (és vele a programot) át kellene alakítani, úgy, hogy képes legyen a népszerűbb kép- és videomegosztó oldalak (mint amilyen a Picasa, ImageShack, vagy a Youtube) tartalmában böngészni. A panelen lehetne egy keresőmező, illetve egy találati lista. A találati listában a keresett kifejezésnek megfelelő kép- vagy videotalálatok előnézeti képei jelennének meg. A felhasználó kiválaszthatna egy képet, ezután a „Beágyazás” gombra kattintva a kiválasztott média linkje hozzáadódik a szerkesztett bejegyzés szövegéhez.

8.6. Portolás mobil eszközökre

Ha hinni lehet a szakértők véleményének, illetve ha megnézzük a piaci változásokat, és a felhasználói igényeket, a hagyományos személyi számítógépek (tehát a fekete vagy fehér színű, hatalmas, zúgó doboz) ideje hamarosan lejár, helyüket a sokkal kisebb, hordozható eszközök, a notebookok, táblagépek, okostelefonok, okostelevíziók vehetik át. Hogy ez valóban így történik-e, nem lehet biztosan tudni, de jobb rá felkészülni. A mandarina jövője ezeken az eszközökön is biztosítható.

A felhasználói felületet igyekeztem úgy megalkotni, hogy a program érintőképernyőkön is használható legyen. A programot 800x480-as felbontásra optimalizáltam. Egy olyan eszközön, amely viszonylag nagyobb méretű kijelzővel rendelkezik, a program a felület megváltoztatása nélkül is futhat.

Kisebb méretű eszközökön (okostelefonok) viszont már módosításra szorul a felület külalakja. Ez hosszabb, de nem lehetetlen munka. Meg kell vizsgálni a képernyőn látható elemeket, melyek maradhatnak kinn a képernyőn mindig, és melyek azok, amelyeket érdemes elrejteni, menüpontok, vagy ikonok mögé.

9. Üzembe helyezés különböző operációs rendszereken

A most következő fejezetben szeretném bemutatni a kész program üzembe helyezési lépéseit. Mivel két platformon (Windows, Ubuntu) fejlesztettem és teszteltem a programot, és mivel ennek a két rendszernek ismerem a program-telepítési mechanizmusát, ezen a két rendszeren szeretném ismertetni az egyes lépéseket.

Mindenekelőtt azonban összegezném, hogy végül milyen komponensekből épül fel a program.

9.1. Szükséges könyvtárak

A program használatához a következő könyvtárak kellenek (verziószámmal együtt):

- QtCore (4.6+)
- QtGui (4.6+)
- QtNetwork (4.6+)
- QtSql (4.6+)
- QtXml (4.6+)
- QtWebkit (4.6+)
- Phonon (4.6+)
- valamint néhány platformtól függő könyvtár:
- Windows rendszer esetén: MinGW
- Linux esetén libstdc++6

9.2. Üzembe helyezés Windows rendszereken

Annak a felhasználónak, aki Windows XP-t, Windows Vista-t, vagy Windows 7-t használ, igazán könnyű dolga van. A program két formátumban érhető el:

- tömörített ZIP állomány
- tömörített önkibontó RAR állomány

Mindkét formátum a programon kívül tartalmazza a szükséges erőforrásokat, könyvtárakat, valamint egy előre elkészített parancsikont.

A ZIP állomány „telepítése” pofonegyszerű. Egy tömörítőprogrammal, amely képes az

ilyen formátumú tömörített állományok kezelésére, a fájlrendszeren belül a kívánt mappába ki kell csomagolni. A program a futtatható mandarina.exe nevű állományra, vagy a parancsikonra duplán kattintva azonnal indítható. Igény szerint a parancsikon kihúzható a munkaasztalra is.

A RAR önkibontó állomány telepítése hasonló, azzal a különbséggel, hogy a kibontáshoz nem szükséges külön archívumkezelő program. Az önkibontó program indítása után meg kell adnunk azt az elérési útvonalat, ahol a program állományait látni szeretnénk. A program indítása ezután ugyanúgy a mandarina.exe fájlra való dupla kattintással történhet.

9.3. Üzembe helyezés Linux operációs rendszereken

A forráskód manuális lefordításával a program gyakorlatilag bármilyen modern A következő lépések bármilyen Linux disztribúción működőképesek.

1.) először is ellenőrizzük, hogy a következő csomagok mindegyike telepítve van-e a rendszerünkön: libqt4-dev qt4-qmake g++ make. Ha valamelyik hiányozna, mindenképpen telepítsük. A csomagok nevei a Debian csomagtárolója szerinti nevek. Más disztribúción értelemszerűen azokat a csomagokat kell telepíteni, amelyek ekvivalensek a fenti csomagokkal.

2.) töltsük le a program honlapjáról a source tarball-t, amely a szokásoknak megfelelően egy tar.gz formátumú tömörített állomány

3.) ezt az állományt tömörítsük ki az általunk kedvelt fájlkezelővel, vagy terminálban a tar paranccsal. Például ha a tömörített állomány neve mandarina-0.4999.tar.gz, a megfelelő parancs a következő: tar xzvf mandarina-0.4999.tar.gz

4.) lépjünk be a létrejött könyvtárba a cd mandarina-0.4999/ paranccsal

5.) adjuk ki a qmake mandarina.pro parancsot

6.) ha ez rendben lefutott, indulhat a program fordítása a make paranccsal

7.) a létrejött binárist a megfelelő helyre másolhatjuk a sudo make install paranccsal

8.) ha nem történt semmiféle hiba, a program a mandarina paranccsal indítható

A telepítés után más dolgunk nincs. A szükséges beállítófájlokat a program az első indításakor létrehozza.

10. Összefoglalás

Szakdolgozatomban egy asztali blogkliens kifejlesztését tűztem ki célul. Célom az volt, hogy egy bárki számára könnyen kezelhető, egyszerű és gyors programot írjak, amely a modern operációs rendszerek mindegyikén képes futni.

A programot C++ nyelven, a Qt keretrendszer segítségével valósítottam meg. A különböző blogmotorokkal való kommunikációhoz az XML-RPC alapú publikációs protokollt használtam fel. Az általam meghatározott célokat (véleményem szerint) kielégítően meg tudtam valósítani, bár a program természetesen még messze van a tökéletestől.

Az alkalmazás jövőbeni fejlődési irányai lehetnek: más publikációs protokollok támogatása, egyszerre több bejegyzés kezelése, a felhasználói felület nem egészen kiforrott részeinek (bejegyzésszerkesztő, képek és videók beágyazása) továbbfejlesztése, illetve megjelenés az új generációs mobil eszközökön. Emellett fontosnak tartom még a program többszálúsítását.

Felhasznált irodalom

[1] http://en.wikipedia.org/wiki/Blog (2011. szeptember-október)

[2] http://hu.wikipedia.org/wiki/Blog (2011. szeptember-október)

[3] http://en.wikipedia.org/wiki/History_of_blogging (2011. szeptember-október)

[4] http://hu.wikipedia.org/wiki/Blog#T.C3.B6rt.C3.A9nelem (2011. szeptember-októ ber)

[5] http://isaacyassar.blogspot.com/2009/03/history-of-blogger.html (2011. szeptember)

[6] http://www.holblogoljak.eoldal.hu/cikkek/a-blog/a-blogolas-tortenelme.html (2011. szeptember)

[7] http://en.wikipedia.org/wiki/Trent_Lott (2011. szeptember)

[8] http://en.wikipedia.org/wiki/XML_RPC (2011. szeptember)

[9] http://openacs.org/api-doc/procs-file-view?path=packages%2flars-blogger%2ftcl %2fblogger-api-procs%2etcl (2011. augusztus)

[10] http://en.wikipedia.org/wiki/Xmlrpc#Examples (2011. szeptember)

[11] http://www.sixapart.com/developers/xmlrpc/movable_type_api/ (2010. augusztus)

[12] http://www.xmlrpc.com/metaWeblogApi (2010. augusztus)

[13] http://en.wikipedia.org/wiki/MetaWeblog (2011. szeptember)

[14] http://www.blogger.com/developers/api/1_docs/ (2010. augusztus)

[15] http://weblogs.asp.net/metablog.ashx (2010. augusztus-szeptember)

[16] http://codex.wordpress.org/XML-RPC_wp (2010. augusztus)

[17] http://en.wikipedia.org/wiki/Qt_(framework) (2011. augusztus)

[18] Jasmin Blanchette, Mark Summerfield: C++ GUI Programming with Qt 4 (2nd Edition)
Prentice Hall, 2008. ISBN: 0-13-235416-0

[19] http://techcrunch.com/2008/01/28/nokia-acquires-trolltech-for-153-million/ (2011. augusztus)

[20] http://www.digia.com/C2256FEF0043E9C1/0/405002251 (2011. július)

[21] http://labs.qt.nokia.com/2011/10/21/the-qt-project-is-live/ (2011. október)

[22] http://qt.nokia.com/qt-in-use/story/customer (2011. július)

[23] http://doc.qt.nokia.com/ (2010-2011)

[24] http://en.wikipedia.org/wiki/Unity_(desktop_environment)#Unity_vs._Unity_2D (2011. október)

[25] http://www.sqlite.org/datatype3.html#boolean (2011. július)

Mellékletek

- egy darab CD-ROM (A melléklet), amely tartalmazza:
    - A program forráskódját
    - Windows rendszereken használható binárisokat, tömörített állomány formájában
- Az XML-RPC alapú publikációs protokoll általam használt függvényeinek listája (Blogger, MetaWeblog, Movable Type, Wordpress) (B melléklet)
- A program részét képezi még egy előre elkészített tesztkörnyezet is, amely valójában egy WordPress alapú blog:

    A blog URL címe: mandarina.blogolj.net

    Felhasználónév: mandarina

    Jelszó: C32xJv8bFK