DIPLOMAMUNKA

LUKÁCS ATTILA

DEBRECEN
2008

Debreceni Egyetem
Informatikai Kar

XML-alapú információmegosztás RSS és Atom formátumokkal

Témavezető:
Jeszenszky Péter
egyetemi adjunktus

Készítette:
Lukács Attila
programtervező matematikus
hallgató

Debrecen
2008

Kivonat

Dolgozatom két formátumról, az RSS-ről és az Atomról, valamint a hozzájuk kapcsolódó protokollokról szól. Látni fogjuk majd, hogy egyszerűségük ellenére milyen jól és mennyire hatékonyan használhatóak, amikor webes együttműködésről, információmegosztásról van szó. Bemutatok néhány, weben alkalmazott API-t és módszert, végül az általam választott ROME keretrendszer segítségével általam készített egyszerű, Apache Tomcaten futó webes szolgáltatást ismertetem.

Tartalomjegyzék

Ábrák jegyzéke

Köszönetnyilvánítás

Ezúton fejezem ki köszönetemet témavezetőmnek, Jeszenszky Péternek, aki észrevételeivel és hasznos tanácsaival segítette munkámat.

1. fejezet

Bevezetés

Manapság nincs az a kezdő internetező, aki ne hallott volna webnaplókról, másnéven a blogokról: a blogírás hihetetlenül népszerű tevékenység lett, hihetetlenül rövid idő alatt. A blogok ma már nem csak a netezők életét befolyásolják, ott vannak mindenhol, gyakran a legnagyobb híroldak, újságok, televíziók is egy blogról értesülnek valamilyen fontosabb történésről. Némelyik blog látogatottsága olyan nagy, hogy írója pusztán a blog írásából (illetve az azon elhelyezett reklámokból) képes megélni. Ezek egyben formálják is olvasóik szemléletét, tehát közvetetten, valamilyen formában az egész világot. Blogot pedig ma már nem csak az írhat, aki felületesebb vagy mélyebb informatikai ismeretekkel rendelkezik, hanem szinte bárki, aki az internethez hozzáfér.

Hosszú ideje nincs az az informatikus, aki nem látja be, milyen nagy hatást gyakorolt az XML megjelenése az informatikára. Az XML segítségével gyakorlatilag bármilyen információhalmazt le lehet írni strukturáltan. Az XML ember és gép által egyaránt egyszerűen olvasható, önmagát dokumentálja, támogatja a Unicode-ot, szabványos, platformfüggetlen, szigorú s ezért könnyen ellenőrizhető nyelvtana van, végül pedig teljesen elfogadott, licenszektől és jogdíjaktól mentesen használható.

Az internetes szakemberek körében gyakran lehet hallani ezt a kifejezést: *Web 2.0*. Furcsának tűnhet, hiszen úgy néz ki, mint egy szoftver verziószáma, holott szó sincs arról, hogy a Webnek (vagy inkább csak kisbetűvel: webnek) újabb vagy régebbi verziója lenne – ez egy felkapott megjelölése annak az újfajta *szemléletnek*, amelyben szinte az összes ma sikeres weboldal osztozik. Ennek a története az évtized elejére nyúlik vissza: a nevezetes dotkom-buborék kidurranásakor rengeteg informatikai céget (főleg kis cégeket, de jónéhány igen nagyot is), weboldalt rántott magával a semmibe. Volt azonban néhány, amely megmaradt, mert ki-

emelkedett népszerűségében, újfajta, izgalmas alkalmazásával. Ezek az oldak bizonyos hasonlóságot is mutattak, közös tulajdonságokkal rendelkeztek. A szakértők egyetértettek abban, hogy ez fordulópontot jelent a web történetében. Példákon keresztül megjelölték, melyek tartoznak az 1.0-hoz, melyek a 2.0-hoz, és hogy ezeket mi különbözteti meg.

Először is, ezek az oldalak bizonyos *szolgáltatást* nyújtanak felhasználóiknak. Minden esetben igaz, hogy *minél többen használják a szolgáltatást, az annál jobb*. (Természetesen, minél többen használják az oldalt, annál több hirdetés értékesíthető rajta.) Ez úgy érhető el, hogy maguk *a felhasználók adják hozzá az értéket* a szolgáltatáshoz azzal, hogy használják: adatokkal látják el. Az igazi „webkettes" szolgáltatások a felhasználók közreműködéséből származó hálózati hatásokat használják ki:

blogok publikálás helyett részvétel, a felhasználók két irányban kommunikálnak, kommenteket is írhatnak egy másik felhasználó naplóbejegyzéséhez

Wikipédia radikális bizalom, az a képtelen ötlet, hogy bárki írhat bármiről, annak reményében, hogy a „több szem többet lát" elven minőségi tartalom állhat elő (*wisdom of crowds*: a tömegek bölcsessége)

BitTorrent radikális decentralizáltság, paradox módon annál gyorsabb a letöltés, minél többen töltenek le – mivel minden letöltő fél egyben feltöltő is[1]

Flickr, del.icio.us címkék használata, nem előre meghatározott taxonómiák rákényszerítése a felhasználókra

eBay, Amazon felhaszálói észrevételek egy-egy termékhez

A Web 2.0-s alkalmazásoknak emelett gyakran közös jellemzőjük a felület fokozott felhasználói élménye (az AJAX technika segítségével), a felhasználók általi „bütykölhetőség" (*hackability*), az örök-béta állapot, az adatok szemcsézett elérhetősége, és számos más közös tulajdonság.

Összefoglalva, a Web 2.0-s szolgáltatások azért lettek sikeresek, mert lehetővé tették felhasználóik számára, hogy *saját adataikat kezeljék* a rendszeren belül, egyrészt egy könnyen kezelhető felülettel az átlagos felhasználók számára, másrészt valamilyen egyszerű API-val az informatikusoknak, amely lehetővé tette, hogy a szolgáltatást ne csak egy meghatározott módon lehessen igénybe venni. Az RSS (és természetesen az Atom is) jó példa egy ilyen – igen egyszerű – API-ra.

[1] A BitTorrent valójában nem webes szolgáltatás, jellege miatt azonban mégis a Web 2.0-hoz sorolják

Az RSS- vagy Atom-feed ma már egyetlen komoly weboldalról sem hiányzik. És nem is csak egyféleképpen használhatók fel. Nagy részben ezeknek a formátumoknak köszönhető a *webkettő* sikere.

Dolgozatom tehát két formátumról, az RSS-ről és az Atomról, valamint a hozzájuk kapcsolódó protokollokról szól. Látni fogjuk majd, hogy egyszerűségük ellenére milyen jól és mennyire hatékonyan használhatóak, amikor webes együttműködésről, információmegosztásról van szó (bizonyos később részletezett okokból elsősorban az Atomra koncentrál majd a dolgozat). Bemutatok néhány, weben alkalmazott API-t és módszert, végül az általam választott ROME keretrendszer segítségével általam készített egyszerű, Apache Tomcaten futó webes szolgáltatást ismertetem.

2. fejezet

Együttműködés a web-en

Mit tehet az internetező, ha kíváncsi, milyen újdonságok vannak kedvenc oldalain? Legáltalánosabban, végigjárja az őt érdeklő weblapokat, újdonságok után kutatva. Ugyanígy, ha egy weboldal tulajdonosa saját oldalán akar megjelentetni a témához illeszkedő weblapok híreit, cikkeit, folyamatosan az érdekesnek ítélt weblapokat kell böngésznie. Hamar felmerült az ötlet, mi lenne, ha az újdonságok, friss tartalmak automatikusan érkeznének meg.

2.1. A kezdetek

Az első próbálkozások között említést érdemel a PointCast. A PointCast vállalat 1992 óta egy olyan szoftvert készített, amely friss híreket jelenített meg a felhasználó képernyőjén, mégpedig egy képernyőkímélőben elhelyezve. A PointCast szerverei folyamatosan küldték a híreket és reklámokat; komoly médiafigyelmet is kapott a termék, amikor 1996-ban elindult. Történetesen a Windows 98-ban is integráltan jelen volt. Újdonságnak számított ez akkortájt, de a népszerűsége elég alacsony maradt, leginkább azért, mert a sávszélességigénye meghaladta az akkoriban elérhető modemes internet kapacitását, és a tartalomhoz képest túl sok reklámot jelenített meg. A felhasználók nem is mindig találták hasznosnak az információkat.

Egy másik jelentős fejlesztés Ramanathan V. Guha, indiai informatikus nevéhez fűződik: 1995 és 1997 között az Apple-nél kidolgozták a Meta Content Framework-öt (MCF), egy olyan formátumot, amely weboldalakról tárolt metaadatokat és más információkat. Amikor kiderült, hogy az Apple nem kívánja folytatni a fejlesztést, Guha átigazolt a Netscape-hez, és ott folytatta. Ekkoriban

találkozott Tim Bray-jel, aki akkortájt az XML kidolgozásán fáradozott, és segítségével XML alapokra ültette át az MCF-et, amelyből később kifejlesztette az RDF-et.

A Microsoft is előrukkolt a fejlesztésével, amely friss infókat szállított a felhasználók desktopjára. Ez volt a Channel Definition Format (CDF), a célja egy weboldal tartalmának és struktúrájának leírása volt. Az elfogadottsága mérhetetlenül kicsi maradt, a lehetőségei szűkösek. Az alapja már ennek a formátumnak is XML volt (bár nem szabványos módon).

1997-ben a Netscape egy újabb technológiát mutatott be, az Aurorát, amely a még be sem fejezett RDF-en alapult. Valamivel később még ebben az évben, Dave Winer egy saját tervezésű XML-alapú formátumot publikált, a ScriptingNews-t, amelyet saját blogján kezdett el használni és ajánlani. (Dave Winerről még hallunk a későbbiekben is. Egyébként az ő blogja egyike a legkorábban megjelent blogoknak.)

2.2. Push vs. pull

Bármilyen adat, amely a felhasználót érdekli vagy a felhasználónak szól, a szervertől a kliens irányába áramlik. Amiben megkülönböztethető az ilyen kommunikáció, az a kezdeményező fél.

Ha a kezdeményező a (központi) kiszolgáló, akkor *push* technológiáról beszélünk. Ez általában feltételez egy előzetes megállapodást, regisztrációt, előfizetést, amelyben a kommunikáció részleteit rögzítik, például hogy hogyan, mikor, és milyen adat küldését kezdeményezheti a kiszolgáló. Tipikusan ilyen alkalmazás például az SMTP, az azonnali üzenetküldők (instant messenger, mint például az ICQ vagy a Jabber-hálózatok) protokollja, vagy a rövidszövegesüzenet-szolgáltatás (SMS) a GSM-hálózatokon. A fenti korai szindikációs protokollok közül ide tartozik a PointCast.

Amennyiben a kommunikáció kezdeményezője az ügyfél, akkor a *pull* technológa áll a kommunikáció mögött. Itt a szolgáltatástól függően szükséges lehet előzetes regisztráció, de nem minden esetben kötelező. A elektronikus levelezésben (szemben az imént említett SMTP-vel) ilyen protokoll a POP3 és az IMAP4, de ide tartozik a HTTP is (a legtöbb esetben, mivel a HTTP-nek is van push-változata). Mivel a – PointCast kivételével – korábban említett technológiák csak formátumok, azok a leginkább a HTTP-re támaszkodnak, vagyis ide sorolhatók.

Figyelemreméltó, hogy a push-hoz hasonló protokoll hogyan váltható ki pull technológiával. Tekintsünk egy email szolgáltatást, amelyben a levelező kiszolgálóhoz POP3-elérést biztosítanak. Ha a levelezőprogram úgy van beállítva, hogy az új leveleket az indítás után is rendszeresen (mondjuk 5 percenként) lekérdezze, akkor a felhasználó úgy tapasztalja, mintha a levelet a kiszolgáló küldené (amennyiben gyakran kap leveleket, és a levelezőprogram folyamatosan fut). Technikailag pull protokoll segítségével így push szimulálható. Hasonló módon, tervezhető olyan üzenetküldő rendszer, amelyben az üzeneteket (rövid üzenetek) a szerver tárolja, és minden kliens rendszeresen (néhány másodperces periódussal) lekérdezi a neki szóló üzeneteket. A felhasználók észre sem veszik, hogy nem a szerver küldi az üzenetet, hanem a kliensprogram kéri el.

2.3. Feed és aggregátor

Abban, hogy push vagy pull technológiát használnak, a szindikációs technikák is eltérnek. Bár, ahogy láttuk, nagy túlsúlyban vannak a potenciálisan pull-alapú szindikációs protokollok. Láthattuk, hogy a PointCast push technikája, bár felkapott lehetőség volt, a felhasználók számára nem volt kellőképp ellenőrizhető – dönteniük kellett, hogy használják a szolgáltatást, vagy teljesen kiszállnak belőle.

A másik lehetőség sokkal vonzóbb lehet a felhasználó számára. A korai szindikációs formátumok nem voltak ugyan egyenértékűek a PointCast-tel, de valamilyen információt hordoztak a weboldalról, amelyhez tartoztak, és igény szerint voltak lekérdezhetőek. Amennyiben változó tartalommal, a változás követhető is lehet. Ez egyenes irányt mutat az RSS felé. Sőt, az RSS és az Atom pontosan ezen az elven működik.

A *feed* (csatorna, hírfolyam – sajnos nem létezik kellőképp elfogadott és elterjedt magyar változat) egy lista, amely frissített tartalomelemeket foglal magában. A feedet általában a tartalom szolgáltatója hozza nyilvánosságra (bár elképzelhető olyan szituáció, amikor nem, például screenscraper futása során). Többnyire XML-alapú dokumentum, amelyben az elemek (leggyakrabban) egy friss hír vagy egy friss naplóbejegyzés egyedi oldalára mutató linket és más metaadatot tartalmaznak. Természetesen a feedet frissíteni kell, amikor friss tartalom jelenik meg.

Egy érdekelt fél (felhasználó vagy más ágens) egy feedre virtuálisan felirat-

kozhat. Ez azt jelenti, hogy egy megfelelő szoftver segítségével rendszeresen lekérdezi az adott feedet, így folyamatosan hozzájut a legfrissebb információkhoz (hírekhez, linkekhez, stb.): ez a szoftver az *aggregátor*. Az elnevezés arra utal, hogy egy aggregátor több feedre is feliratkozhat, így azok tartalmát összefésülheti, egységesen jelenítheti meg, vagy szűrheti is. Sőt, nem ritka, hogy egy aggregátor több feed összefésüléséből újabb feedet hoz létre, amely aztán ismét publikálható.

Ezt az architektúrát összehasonlítva egy hírlevél-szolgáltatással, megállapíthatjuk, hogy míg a hírlevél esetében meg kell adni a szolgáltatónak emailcímünket (vagyis előzetes megállapodás szükséges, ahogy egy push technológiánál – minthogy az is, hiszen SMTP-t használnak), addig a feedek esetében semmilyen előzetes információcsere nem szükséges (leszámítva, hogy tudnunk kell a feed helyét). Az előfizetés a hírlevél esetében a szolgáltatónál történik, míg a feed esetében az aggregátorban jegyezzük be, hogy hogyan érhető el a feed. Ez HTTP esetében egy URL feljegyzését jelenti. Nem mellesleg, mivel nem osztjuk meg emailcímünket, megkímélhetjük magunkat a spamtől. Ha pedig már nem kívánunk értesülni a frissítésekről, nem kell leiratkozni: egyszerűen eltávolítjuk az URL-t az aggregátorból.

2.4. Sokoldalú feed

Bár maga a feed nem több, mint egyszerű elemek sorozata, a felhasználási lehetőségek száma meglepően magas. Valószínűleg össze sem lehetne számolni, mert az aggregátorok nagyon sokfélék lehetnek.

2.4.1. Webes újdonságok

A legegyszerűbb lehetőség a blogokon elhelyezett feedek, ezek a legújabb naplóbejegyzéseket tartalmazzák, vagy legalábbis a bejegyzés címét, a létrehozás és/vagy a módosítás dátumát, egy részletet a bejegyzés szövegéből vagy magát az egész szöveget, és egy URL-t a bejegyzés egyedi lapjára. Ezen kívül az egész oldalra vonatkozó információkat is tartalmaznak, például az oldal címét, a létrehozó nevét és emailcímét, az oldal leírását és URL-jét, nyelvét, stb. Itt emlékeztetem az olvasót, hogy legkorábban, 2001 táján a blogok voltak az első bevezetői a feedeknek. Egy másik alkalmazás a hírportáloknál jelent meg, a friss híreket hozták ebben a formában is nyilvánosságra. Azonnal belátható azonban, hogy

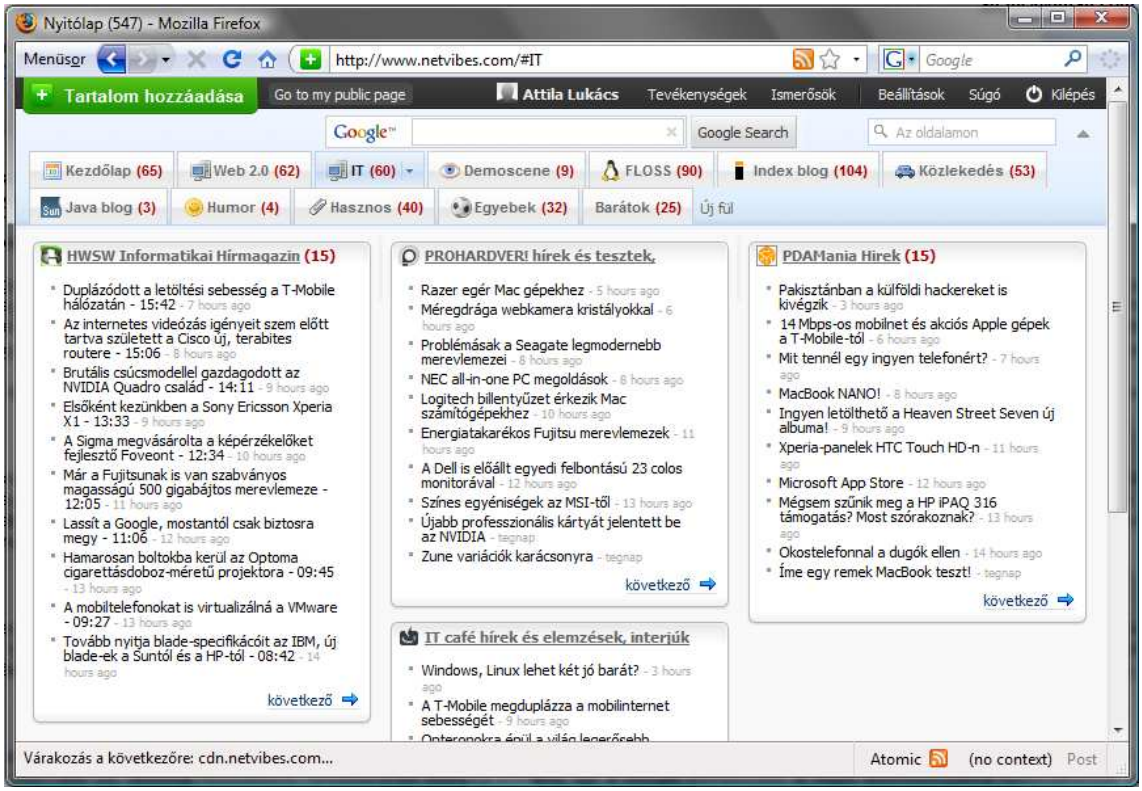
bármilyen híreket, újdonságokat nyilvánosságra hozó weboldalon használható a technológia.

Feedet bármihez létre lehet hozni. A blogoknál például meg lehet bolondítani a képletet azzal, hogy feedet adhatunk minden bejegyzéshez kapcsolódóan a hozzászólásokhoz, vagy az összes hozzászóláshoz együttesen. Egy blogon belül adhatunk feedet az összes címkéhez, egyedileg, amelyben csak az adott címkéhez kapcsolódó friss bejegyzések szerepelnek. Sőt, újabban keresési eredményeket is el lehet érni feeden keresztül – ennek a feldolgozására már csak megfelelő aggregátort kell találni. (Például, egy aggregátor több feedet is összefésülve, azokat a keresési paraméterrekkel elérve, több keresőmotorból származó adatot szolgáltat egy helyen.)

Mivel a feedeket az aggregátorok képesek feldolgozni, így a megjelenítésről is az aggregátorok gondoskodnak. Az aggregátor lehet egy asztali kliensprogram, ekkor általában valamilyen hírolvasásra alkalmas felülettel rendelkezik. A bejegyzett feedeket külön vagy összefésülve kezeli, és a feed egy elemének kiválasztásakor megjeleníti a feedelemben tárolt információkat vagy a mutatott URL alatt található weboldalt. Ma már több népszerű levelezőprogram is támogatja a feedeket, és a felület tökéletesen alkalmas is a megjelenítésre, ha például egy hírlevél-szolgáltatást feeddel helyettesítünk. A böngészőprogramok is (már az Internet Explorer is) támogatják a feedeket, például a Mozilla Firefox úgynevezett „intelligens könyvjelzők" formájában. Létezik olyan aggregátor, amely a képernyő kis területén, széles csíkban jeleníti meg a regisztrált feedekben megjelenő híreket, összefoglalójukat vagy csak címüket futó szövegként.

Az asztali és felhasználó-oldali klienseken túl léteznek úgynevezett feedolvasó szolgáltatások, amelyek a böngészőprogramban jelennek meg, és nem igényelnek natív asztali klienst. Természetes környezetet jelent ez, hiszen a böngészőprogram azonnal adott a weboldalak megjelenítéséhez. Ilyen szolgáltatás például a Google Reader és Bloglines. Más szolgáltatásokkal, például a Netvibes, az iGoogle, a live.com vagy a Pageflakes használatával gyakorlatilag egy személyes oldal állítható össze, amelyen megtalálható a számunkra érdekes összes olyan weboldal legfrissebb összefoglalója, amely valamilyen formában feedben szállítja a legrissebb tartalmakat. (Ezek az odalak egyébként különböző minialkalmazások elhelyezését teszik lehetővé saját oldalunkon, ezek között van a feedolvasó.) Az online feedolvasók túlnyomó többsége AJAX-ot használ az oldal kinézetéhez és a tartalom frissítéséhez.

A feedekből a webmesterek is profitálhatnak, mert oldalaikon automatikusan



2.1. ábra. A Netvibes feedolvasója a webböngészőben

jelenhetnek meg eredetileg más oldalakon publikált friss hírek, így valódi webes együttműködés jön létre a két oldal között.

2.4.2. Speciális feedek

A feedek nem csak weboldalak tartalmának összefoglalására és az új tartalmakról szóló értesítésekre használható; pontosabban fogalmazva, nem csak weblapok lehetnek friss tartalmak. Az RSS-ben az `enclosure` tag megjelenésével lehetővé vált multimédiás állományok feedelemekhez csatolására, és ez közvetlenül elsősorban a *podcasting*, a fotófeedek és a *vodcasting*, *vlogging* megjelenéséhez vezetett. A podcasting azt jelenti, hogy a feedelemekben a weboldalra mutató link helyett egy zenei állományra mutató URL jelenik meg, így az a feed frissülésekor letölthető, és egy megfelelő lejátszó csatlakoztatásakor a számítógéphez, arra automatikusan áttölti az aggregátor. Ezt a módszert először iPod-oknál használták – a neve innen származik –, és ma már számtalan alkalmazása ismert: ilyen módon terjesztenek rádióműsorokat, zenei összeállításokat, újságok interjúit vagy oktatási segédanyagokat. A vodcasting ennek a videókra értelmezett változata.

A fotófeedekhez csak látogassunk el például a `www.flickr.com`-ra vagy a Google Picasaweb szolgáltatásának oldalára.

Érdekes jelenség, hogy újabban már BitTorrent-kliensekbe is beépítik a feedek támogatását (melyek így aggregátorrá válnak). Ezek a kliensek alapvetően nem HTTP-re építenek, de .torrent kiterjesztésű állományokat használnak egy adott file vagy file-csoport elérésének leírására. Amint azonban automatikusan férhetnek hozzá a weben keresztül a .torrent állományokhoz, ez azt jelenti, hogy az általuk címzett adatokhoz is automatikusan férhetnek hozzá. Így ebből a webes lehetőségből egy alapvetően nem webes alkalmazás is tudott profitálni.

3. fejezet

Szindikációs formátumok

3.1. RSS

Mint láttuk, számos próbálkozás előzte meg az RSS-t, de valódi áttörést csak ez a formátum hozott. Az RSS valójában egy egész formátumcsaládot jelent, amelyek megpróbálják ugyanazt a célt szolgálni: frissített tartalmat szállítani, és amelyek gyakorlatilag sajnos kivétel nélkül inkompatibilisek egymással.

Eredetileg Ramanathan V. Guha fejlesztette ki a Netscape-nél, a MyNetscape-pel való használathoz. 1999 márciusában hozták nyilvánosságra, ez volt a 0.90-es verzió, amely a Resource Description Framework (RDF) tervezetére épült, de végleges kiadásával már nem volt kompatibilis.

3.1.1. RSS 0.90

Az RSS az RDF Site Summary rövidítése. A következő példafeed a hivatalos specifikációból származik [4].

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://channel.netscape.com/rdf/simple/0.9/"
>
  <channel>
    <title>Mozilla Dot Org</title>
    <link>http://www.mozilla.org</link>
    <description>the Mozilla Organization web site</description>
  </channel>
  <image>
    <title>Mozilla</title>
    <url>http://www.mozilla.org/images/moz.gif</url>
    <link>http://www.mozilla.org</link>
```

```
15  </image>
    <item>
      <title>New Status Updates</title>
      <link>http://www.mozilla.org/status/</link>
    </item>
20  <item>
      <title>Bugzilla Reorganized</title>
      <link>http://www.mozilla.org/bugs/</link>
    </item>
    <item>
25    <title>Mozilla Party, 2.0!</title>
      <link>http://www.mozilla.org/party/1999/</link>
    </item>
    <item>
      <title>Unix Platform Parity</title>
30    <link>http://www.mozilla.org/build/unix.html</link>
    </item>
    <item>
      <title>NPL 1.0M published</title>
      <link>http://www.mozilla.org/NPL/NPL-1.0M.html</link>
35  </item>
  </rdf:RDF>
```

Mint az látszik, jólformált XML dokumentumról van szó, XML prológussal. Ezután a gyökérelem az `rdf` névtérből származó `RDF`, mint minden RDF-dokumentumnak. Ennél több köze nincs az RDF-hez, közvetlenül ezen elem gyermekeiként szerepelnek `channel`, `image` és `item` elemek. A `channel` elem írja le a feedet, a `title`, `description` és `link` elemek tartalma rendre a csatorna nevét, hosszabb szöveges leírását és az oldal URL-jét adja meg. `channel` elemből egyetlen szerepelhet az állományban, megadása kötelező. Az `image` elem a csatornához tartozó ikont jelöli, gyermekei a `title`, `url` és `link`, amelyek az ikon címét, forrásának URL-jét és az oldalra mutató linket képviseli. Ezután néhány `item` elem következik, amelyben a `title` és `link` egy címet és URL-t ad meg. A specifikáció itt megenged egy további `textinput` elemet, benne `title`, `description`, `name` és `link` elemekkel, ezt tetszőleges adat visszaküldésére lehet használni, HTTP GET metódus használatával.

Az RSS 0.90 számos megszorítást tesz a tartalomra. Először is, a file egésze nem lehet nagyobb 8 kilobyte-nál, és 15 feedelemnél nem tartalmazhat többet. Karakterkészlet tekintetében egyedül az ASCII-t támogatja, de használhatók a HTML-ben megengedett karakterentitás-hivatkozások; más karakterkészlet nem használható. A szövegben sehol sem használhatók a `'`, `"`, `&`, < és > karakterek, ezeket mind a megfelelő HTML entitásra kell lecserélni. Különösen az `url` és `link` elemek tartalmában kell odafigyelni a `&` karakterre, amennyiben dinamikus lapnak adunk át GET-paramétert. Azokon az elemeken belül, amelyek URL-t tartal-

maznak, csak a `http://` és az `ftp://` előtag megengedett, a többi érvénytelennek minősül (így a `https` is).

Ez a változat még nem igazán tekinthető szindikációs formátumnak, inkább csak egy oldal összefoglalójának.

3.1.2. RSS 0.91

A Netscape 1999 júliusában kiadta az RSS következő verziójának leírását, egyrészt megszabadítva az RDF csomagolástól, másrészt jelentősen kibővítve a Dave Winer által kifejlesztett ScriptingNews formátum néhány elemével. Így olyan hasznos információk kerültek bele, mint az egyes elemek megjelenésének és frissítésének dátuma, nyelv információ, kontakt információk. A Netscape egyúttal át is nevezte a formátumot *Rich Site Summary*-ra. Fő fejlesztője Dan Libby.

Az RSS 0.91 már megköveteli a feedtől, hogy nem csak jólformált, hanem érvényes XML is legyen. Ennek érdekében kötelezővé teszik a dokumentumtípus-deklaráció használatát, hogy DTD-vel érvényesíthető legyen. Nincsenek a formátumban vegyes tartalmú elemek, vagyis egy elem tartalma csak más elemek, vagy szöveges adat lehet.

Ebben a formátumban az XML-fa gyökere az `rss` elem, amelynek egyetlen gyermeke egyetlen `channel` lehet. Attribútum csak egy van, ez pedig az `rss` elem `version` attribútuma, értéke kötelezően `0.91`. A formátum további részletei az alábbi felsorolás szerint alakulnak:

- `<rss>` pontosan 1, a feed gyökere
  - `<channel>` pontosan 1, egy feedben egy csatorna lehet
    - `<description>` pontosan 1, a csatorna leírását tartalmazza
    - `<language>` pontosan 1, a csatorna nyelvét jelöli
    - `<link>` pontosan 1, arra a weboldalra mutat, amelynek a tartalmát leírja
    - `<title>` pontosan 1, a csatorna címe
    - `<copyright>` legfeljebb 1, szabadszöveges, nem ellenőrzött
    - `<docs>` legfeljebb 1, olyan URL, amelyen a csatorna leírása megtalálható
    - `<image>` legfeljebb 1, egy képet vagy ikont rendel a csatornához
      - `<url>` pontosan 1, a kép forrása
      - `<link>` pontosan 1, a kép hivatkozása (az oldalra)
      - `<title>` pontosan 1, a kép címe

    - <description> legfeljebb 1, a kép leírása
    - <width> legfeljebb 1, szélesség
    - <height> legfeljebb 1, magasság
  - <item> tetszőleges sok, a csatornán elhelyezett elemek
    - <title> pontosan 1, az elem címe
    - <link> pontosan 1, az elem URL-je
    - <description> legfeljebb 1, az elem leírása
  - <lastBuildDate> legfeljebb 1, az utolsó módosítás dátuma
  - <managingEditor> legfeljebb 1, az összeállításért felelős személy emailcíme
  - <pubDate> legfeljebb 1, publikálás dátuma
  - <rating> legfeljebb 1, figyelmen kívül hagyott
  - <skipDays> legfeljebb 1, lista a nem figyelt napokról
    - <day> legalább 1, egy napot jelöl
  - <skipHours> legfeljebb 1, lista a nem figyelt órákról
    - <day> legalább 1, egy órát jelöl
  - <textinput> tetszőleges sok, feedback célra
    - <title> pontosan 1
    - <link> pontosan 1
    - <description> pontosan 1
    - <name> pontosan 1
  - <webMaster> legfeljebb 1, az oldalért felelős személy emailcíme

Az XML-ből következően HTML-tartalmak továbbra sem megengedettek a tartalomban, és minden tiltott karaktert a megfelelő entitásra kell cserélni (mint a korábbi kiadásban). A támogatott karakterkódolások száma jelentősen kibővült, messze nem teljes, de már támogatja az ISO-8859 sorozat több tagját és az UTF-8-at. A nyelvek jelölésére bevezették a language elemet, amelynek tartalma a specifikációban megadott listából lehet egy elem (az angol változatain kívül elsősorban európai nyelveket tartalmaz a lista). Több helyen megadható dátum, de ennek formátumára nincs megkötés (a hivatalos példa az RFC 822, az ARPA Internet szöveges üzenetek formátumátumáról szóló 1982-es standard szerinti formátumot használja).

UserLand RSS 0.91

2000 áprilisában az America Online által irányított Netscape áttervezte a My-Netscape-et, eltávolította az RSS-támogatást és a DTD-t az oldalról. Ez gyakorlatilag tulajdonos nélkül hagyta a formátumot, ami később kaotikus verziózáshoz és további inkompatibilitáshoz vezetett. Dave Winer ragadta magához az irányítást, és a UserLand weboldalán publikálta az RSS 0.91 módosított változatát. Lényegében új verzióról van szó, azonos verziószámmal! Technikailag abban különbözik a Netscape verziójától, hogy nem használ DTD-t, így azonban a HTML-ben szokásos entitások sem használhatók, amelyet a Netscape DTD-je megengedett, vagyis annak egy részhalmaza lett a UserLand RSS 0.91. Ennek szerzői jogait Winer magáénak tulajdonította. A UserLand specifikáció kimondja, hogy az RSS már nem egy rövidítés, csak egy név.

Néhány formátumváltozás is történt ebben a verzióban. Winer a `textInput` elemet használja a `textinput` helyett, valamint a `skipHours`-on belül a `hour` a 0–23 tartomány helyett az 1–24 tartományból vehet fel értéket.

3.1.3. RSS 1.0

A Netscape megrendülése után egy másik csoport, az RSS-Dev csoport is megmozdult. Az eredeti 0.90-es specifikációt alapul véve egy új változatot hoztak létre a formátumból. Az RDF-alapú vonalat vitték tovább, de már sokkal modulárisabban felépítve. Például itt vezették be külső szótárak, többek között a Dublin Core használatát. Az RSS ismét RDF Site Summary-t jelent! A csoport állítása szerint kompatibilis a 0.90-es verzióval, ez azonban féligazság: a régi elemek is egy új névtérben vannak.

A dokumentum gyökere itt is `rdf:RDF`, amelynek gyermekei egy `channel` és több `image`, `item` és `textinput` elem lehet. Ezek mindegyikét azonosító URI-val kell ellátni a `rdf:about` attribútum segítségével. Az `item`-ben a korábban létező `title` és `link` mellett megjelenik a 0.91-ben bevezetett `description` elem is, ebben leírás adható meg. A kép, a textinputok és az itemek nem tartoznak automatikusan a csatornához, ehhez a `channel`-en belül meg kell adni az `image`, `textinput`, `items` elemeket. Az `image` és `textinput` elemeken belül az `rdf:resource` attribútummal kell azonosítani, hogy melyik külső szinten megadott `image` és `textinput` *erőforrás* tartozik a csatornához, illetve az `items` elemen belül `rdf:Seq` segítségével szabványos RDF módszerrel kell felsorolni a csatornához tartozó elemeket, konkrétan `rdf:li` elem `rdf:resource` attribútumával megadni

az item azonosítóját (URI).

A fenti tömör leírást szemlélteti az alábbi példa, a hivatalos specifikációból [4].

```
<?xml version="1.0"?>
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns="http://purl.org/rss/1.0/"
>
  <channel rdf:about="http://www.xml.com/xml/news.rss">
    <title>XML.com</title>
    <link>http://xml.com/pub</link>
    <description>
      XML.com features a rich mix of information and services
      for the XML community.
    </description>
    <image rdf:resource="http://xml.com/universal/images/xml_tiny.gif" />
    <items>
      <rdf:Seq>
        <rdf:li resource="http://xml.com/pub/2000/08/09/xslt/xslt.html" />
        <rdf:li resource="http://xml.com/pub/2000/08/09/rdfdb/index.html" />
      </rdf:Seq>
    </items>
  </channel>
  <image rdf:about="http://xml.com/universal/images/xml_tiny.gif">
    <title>XML.com</title>
    <link>http://www.xml.com</link>
    <url>http://xml.com/universal/images/xml_tiny.gif</url>
  </image>
  <item rdf:about="http://xml.com/pub/2000/08/09/xslt/xslt.html">
    <title>Processing Inclusions with XSLT</title>
    <link>http://xml.com/pub/2000/08/09/xslt/xslt.html</link>
    <description>
     Processing document inclusions with general XML tools can be
     problematic. This article proposes a way of preserving inclusion
     information through SAX-based processing.
    </description>
  </item>
  <item rdf:about="http://xml.com/pub/2000/08/09/rdfdb/index.html">
    <title>Putting RDF to Work</title>
    <link>http://xml.com/pub/2000/08/09/rdfdb/index.html</link>
    <description>
     Tool and API support for the Resource Description Framework
     is slowly coming of age. Edd Dumbill takes a look at RDFDB,
     one of the most exciting new RDF toolkits.
    </description>
  </item>
</rdf:RDF>
```

Az RSS ezen változatából kimaradtak, ismét hiányoznak a szindikációhoz elengedhetetlen dátumadatok, viszont hozzáadhatóak külső szótárak, például a Dublin Core segítségével. Ezeket a szótárakat a specifikáció moduloknak nevezi.

Egy modul hozzáadásához a feed (mint XML dokumentum) gyökerében el kell helyezni a megfelelő névtérdeklarációt (például „`xmlns:dc=...`”), majd ezután használható a dokumentumban azon elemen belül, amelynek tartalmára vonatkozik az adat (például `feed`-en belül `dc:date`, `dc:creator` vagy `dc:rights`).

3.1.4. RSS 0.92-0.94

Az RSS 1.0 kiadása után 16 nappal Winer kiadta az RSS 0.92-es változatát, amely a UserLand RSS 0.91-et veszi alapul. Ez a verzió opcionálissá teszi a csatorna `language` és a feedelemek összes elemét. Az `item` elem alatt bevezeti a `source`, a `category` és az `enclosure` elemeket, valamint a `channel` elem alatt a `cloud` elemet. Fontos változás, hogy a `description` tartalommodellje sima szövegről HTML-re változik.

Working draft formában megjelenik még az RSS 0.93 és 0.94. Az RSS 0.93 több `enclosure` elemet is megenged az `item` alatt, és bevezeti az `expirationDate` elemet. Az RSS 0.94 aztán ez utóbbit elveti, és vezeti a `description` elemen belül a `type` attribútumot, hogy meghatározható legyen, milyen formátumot követ a tartalom (sima szöveg vagy HTML). Az RSS 0.93 és 0.94 soha nem lett kiadva, sőt, a 0.94-es verzió leírása nem is található meg, mert az a 2.0-ra mutat, annak ellenére, hogy tudjuk, hogy van eltérés a két változat között. [3]

3.1.5. RSS 1.1

Az 1.0-s változat továbbgondolása egy független fejlesztőtől, egyetlen csoport, szervezet sem támogatta, az RSS-Dev sem.

3.1.6. RSS 2.0

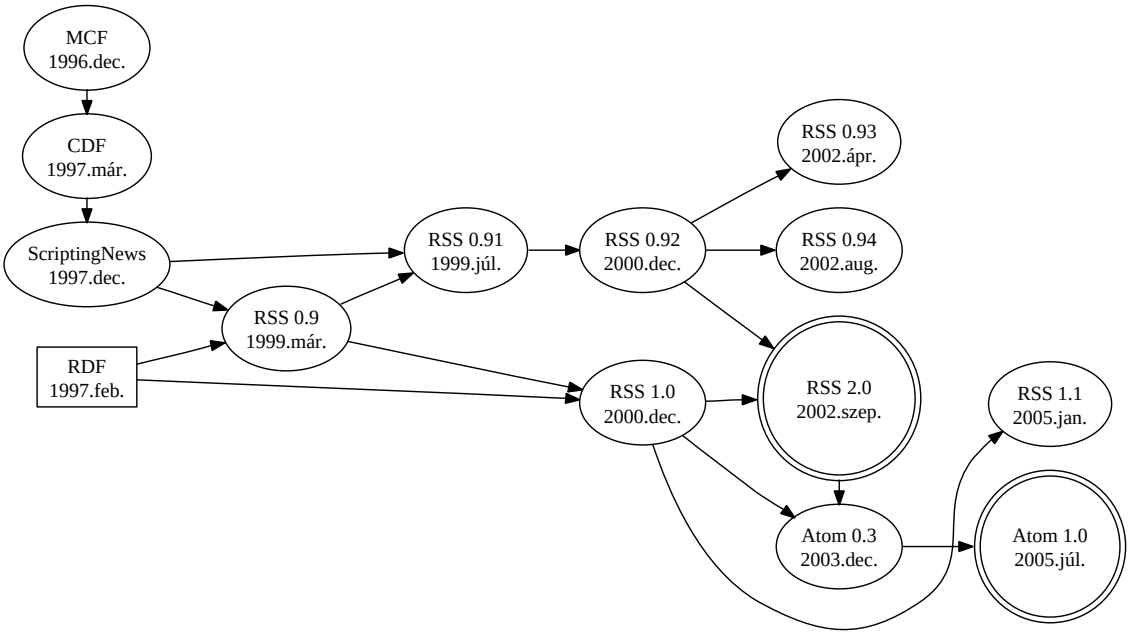
2002 augusztusában végül megszületik Winertől az RSS 2.0, amely *állítása szerint* megegyezik a 0.94-es verzióval. Az RSS jelentése itt már *Really Simple Syndication*. Winer eltávolítja a `type` attribútumot, a `channel` alatt bevezeti a `ttl`, az `item` alatt a `author`, `category`, `comments`, `guid` elemeket. Az RSS 2.0 lehetővé teszi a specifikációban nem definiált elemek használatát, ha azok egy definiált névtérben vannak elhelyezve. A példa eredeti forrása a [4].

```
<?xml version="1.0"?>
<rss version="2.0">
  <channel>
```

```
    <title>Liftoff News</title>
    <link>http://liftoff.msfc.nasa.gov/</link>
    <description>Liftoff to Space Exploration.</description>
    <language>en-us</language>
    <pubDate>Tue, 10 Jun 2003 04:00:00 GMT</pubDate>
    <lastBuildDate>Tue, 10 Jun 2003 09:41:01 GMT</lastBuildDate>
    <docs>http://blogs.law.harvard.edu/tech/rss</docs>
    <generator>Weblog Editor 2.0</generator>
    <managingEditor>editor@example.com</managingEditor>
    <webMaster>webmaster@example.com</webMaster>
    <item>
      <title>Star City</title>
      <link>http://liftoff.msfc.nasa.gov/news/2003/news-starcity.asp</link>
      <description>How do Americans get ready to work with Russians aboard
        the International Space Station? They take a crash course in
        culture, language and protocol at Russia's
        &lt;a href="http://howe.iki.rssi.ru/GCTC/gctc_e.htm"&gt;Star
        City&lt;/a&gt;.
      </description>
      <pubDate>Tue, 03 Jun 2003 09:39:21 GMT</pubDate>
      <guid>http://liftoff.msfc.nasa.gov/2003/06/03.html#item573</guid>
    </item>
    <item>
      <description>Sky watchers in Europe, Asia, and parts
        of Alaska and Canada will experience a &lt;a
        href="http://science.nasa.gov/headlines/y2003/30may_solareclipse.htm"&gt;
        partial eclipse of the Sun&lt;/a&gt; on Saturday, May 31st.
      </description>
      <pubDate>Fri, 30 May 2003 11:06:42 GMT</pubDate>
      <guid>http://liftoff.msfc.nasa.gov/2003/05/30.html#item572</guid>
    </item>
    <item>
      <title>The Engine That Does More</title>
      <link>http://liftoff.msfc.nasa.gov/news/2003/news-VASIMR.asp</link>
      <description>Before man travels to Mars, NASA hopes to design new
        engines that will let us fly through the Solar System more quickly.
        The proposed VASIMR engine would do that.
      </description>
      <pubDate>Tue, 27 May 2003 08:37:32 GMT</pubDate>
      <guid>http://liftoff.msfc.nasa.gov/2003/05/27.html#item571</guid>
    </item>
  </channel>
</rss>
```

3.2. Atom

2003. június 16-án Sam Ruby szoftverfejlesztő lejegyzett blogjában egy rövid gondolatsort arról, hogy szerinte mitől lesz egy bejegyzés megfelelően leírt [8]. A bejegyzés szinte lavinát indított el, hozzászólásaikban sokan csatlakoztak Ruby-hoz, további ötletekkel is szolgáltak. Kicsit később elindítottak egy wiki oldalt,



3.1. ábra. A szindikációs formátumok fejlődése

ahol tovább gyűjtötték az ötleteket. Ezzel kezdődött az Atom születésének folyamata.

Az Atom az RSS-hez hasonlóan egy webes szindikációs nyelv, az XML alkalmazása. Az Atom az RSS számos problémás részét megpróbálja kiküszöbölni, például sokkal részletesebben rögzíti a részleteit, többek között a dátumformátumot, szabályozza a hordozható tartalmat, részletesebben írja le a bejegyzéseket. Számos más előnye is van az RSS-sel szemben. Az Atom az Internet Engineering Task Force (IETF) által beiktatott szabvány, rendelkezik bejegyzett MIME-típussal. Csak két verziója létezik, és a végleges verziót ajánlják használatra.

A 3.1. ábra a formátumok fejlődését mutatja [10].

3.2.1. Atom 1.0

Az Atom formátumában leírt összes elem a `http://www.w3.org/2005/Atom` névtérben van. Egy Atom formátumú dokumentum kétféle gyökerű lehet, vagy a `atom:feed`, vagy az `atom:entry` elemmel kezdődik. A formátumban minden elemen belül megengedett az `xml:lang` és az `xml:base` attribútumok használata, amelyek rendre a nyelvet és azt az URI-t határozzák meg, amelyhez képest a relatív URI-k feloldása történik. Az Atom definiál néhány konstrukciót, amelyet különböző helyeken lehet használni a dokumentumban, ezek szöveget, dátumot

vagy személyt írnak le.

Konstrukciók

A szöveget illetve tartalmat leíró konstrukció célja, hogy meghatározható legyen, mit is hordoz a dokumentumban egy adott elem. A legtöbb elemben ez használható. A hordozott tartalom típusát az adott elem `atom:type` attribútuma határozza meg, melynek értéke `text`, `html` vagy `xhtml` lehet. A `text` és `html` esetében az elem csak szöveget tartalmazhat, más elemet nem, és az XML által szövegben nem megengedett karaktereket szöktetni kell. A különbség természetesen a tartalommodellben van[1]. Az `xhtml` esetében az elem gyermeke egy `xhtml` névtérből származó `div` lesz, amelyen belül szabadon megadható az XHTML tartalom.

A dátum konstrukció arra szolgál, hogy jól definiált módon egységesítse a dátumleírást. Ennek megfelelően megköveteli a dátumok ISO-8601 szabvány szerinti leírását. Ez az egységes megjelenés mellett azért is előnyös, mert a W3C XMLSchema is ezt használja, így egy erre épülő validátor képes ellenőrizni érvényességét.

A személyt leíró konstrukció három gyermekelemmel rendelkezik: az `atom:name`, `atom:uri` és `atom:email` elemekkel, amelyek a személy nevét, weboldalának címét és emailcímét adják meg tartalmukban. Ez a konstrukció megengedi a más szótárból származó elemekkel való kiterjesztését is.

Törzselemek

Az `atom:feed` elem rendelkezik kötelező, ajánlott és opcionális gyermekelemekkel. Kötelező megadni az `atom:id`, `atom:title` és `atom:updated` elemeket. Ajánlott elemei legalább egy `atom:author` és legalább egy `atom:link` elem. Opcionálisan megadható egy vagy több `atom:category`, `atom:contributor`, egy `atom:generator`, `atom:icon`, `atom:logo`, `atom:rights` és `atom:subtitle` elem. Az `atom:feed` szabadon kiterjeszthető.

Az `atom:entry` egy feedelem leírására szolgál. Az `atom:feed`-hez hasonlóan ennek is kötelező megadni az `atom:id`, `atom:title` és `atom:updated` elemeket. Legalább egy `atom:author` és egy `atom:link` itt is javasolt, valamint egy `atom:content` és/vagy `atom:summary` elem, hogy tartalmat is hordozzon. Opcionáli-

[1] A tartalommodell megadását az RSS előbb megoldja, majd a megoldást elveti. Pedig ez egyáltalán nem triviális: például a „<” szöveget text esetén `<`, html esetén `&lt;` sorozattal kell leírni.

sak az atom:category, atom:contributor, atom:published, atom:source, atom:rights elemek megadása (az atom:published és atom:source kivételével mindegyikből több is lehet).

Az atom:content elem hordozza a tartalmat. Ezt úgy határozták meg, hogy szinte bármilyen tartalmat képes legyen szállítani. Ennek megfelelően egyrészt a szöveg konstrukció szabályai érvényesek rá, másrészt típusaként a atom:type attribtumban megadható MIME-típus. Ha ez a típus „/xml"-re vagy „+xml"-re végződik, akkor az elem gyermeke tetszőleges XML tartalom lehet, ha az megfelel a MIME-típusnak. Ha nem XML-tartalomról van szó, akkor Base-64 kódolásban kell tárolni a tartalmat, vagy megadható az atom:src attribútum, amely URI-val jelöli ki a hálózaton a tartalmat, ebben az esetben lehet üres ez az elem.

Metaadat-elemek

Ezek az elemek az atom:feed vagy atom:entry tartalmát írják le, a nevük általában önmagukért beszélnek. Az atom:author és atom:contributor elemek személykonstrukciók, szülőelemük készítőit jelölik. Az atom:title, atom:subtitle, atom:summary, atom:rights elemek szövegkonstrukciók. Az atom:updated és atom:published elemek dátumkonstrukciók. A fenti, máshogy nem meghatározott elemek saját leírással rendelkeznek.

Az atom:id elem az egyik legalapvetőbb elem, amely biztosítja, hogy egy feed vagy feedelem azonosítható legyen, mert univerzálisan egyedi azonosító megadását követeli meg, amely a feed vagy feedelem esetleges változása esetén is megmarad. Tartalma egy URI lehet. Két ID összehasonlításánál számít a kisbetű-nagybetű különbség és az URI % jellel való szöktetése is. Olyannyira komolyan veszi a specifikáció az azonosítást, hogy ha két különböző dokumentumban különböző helyeken azonos ID-t detektál az aggregátor, lépéseket tehet az eredeti elem kiderítésére, mielőtt értesíti a felhasználót.

Az atom:link egy másik nagyon fontos elem, számos más elemen belül megjelenhet, és rengeteg attribútuma van, úgy mint atom:href, atom:rel, atom:type, atom:hreflang, atom:title és atom:length, ezek rendre az URI-ját, a szülőjével mint elemmel való viszonyát, a mutatott erőforrás MIME-típusát, nyelvét, címét, és hosszát határozzák meg. Az atom:rel értéke alternate, related, self, enclosure illetve via lehet.

Az atom:icon és atom:logo a vizuális reprezentációt hivatottak kifejezni, tartalmuk URI lehet. Az atom:generator a feedet kiszolgáló szoftverkomponenst

jelöli, két attribútuma az `atom:uri` és `atom:version`, tartalmuk szöveges lehet.

Az `atom:category` szolgál a feedelem besorolására, három atribútuma, az `atom:term`, `atom:scheme` és `atom:label`. Az első megadása kötelező, ez a besorolás vagy kategória megnevezése. A második egy URI, amely megjelöli a kategorizálás módját, a harmadik pedig egy ember által olvasható szöveges leírást ad (rendszerint megegyezhet az elsővel).

Végül `atom:source` használható arra, hogy amikor egy már meglévő feed olvasásával hozunk létre egy másikat, megőrizzük az eredeti adatait is, amíg már a sajátot továbbítjuk. Ez az `atom:entry` elemen belül jelenhet meg. A teljesség kedvéért lehetséges gyermekelemei: `atom:author`, `atom:category`, `atom:contributor`, `atom:generator`, `atom:icon`, `atom:id`, `atom:link`, `atom:logo`, `atom:rights`, `atom:subtitle`, `atom:title`, `atom:updated`. Ennek az elemnek a kiterjesztése megengedett (minthogy az `atom:entry`-nek is).

A formátum kiterjesztése

Az Atom szabadon kiterjeszthető tetszőleges XML-jelöléssel azokon a pontokon, ahol ez megengedett. Bármilyen elem, amely az Atom szótárában nincs benne, idegen jelölésnek tekintendő. A specifikáció megköveteli, hogy az Atomot fogyasztó feedproceszorok ne álljanak le hibával, ha olyan helyen észlelnek idegen jelölést, ahol használható. Ezek a helyek az `atom:feed`, `atom:entry`, `atom:source` elemekben és a személyt leíró konstrukcióban vannak.

A verziókövetés új névtérrel történhet a jövőben.

3.2.2. Példa egy Atom feedre

A példa forrása a [13].

```
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Example Feed</title>
  <subtitle>A subtitle.</subtitle>
  <link href="http://example.org/feed/" rel="self"/>
  <link href="http://example.org/"/>
  <updated>2003-12-13T18:30:02Z</updated>
  <author>
    <name>John Doe</name>
    <email>johndoe@example.com</email>
  </author>
  <id>urn:uuid:60a76c80-d399-11d9-b91C-0003939e0af6</id>
  <entry>
    <title>Atom-Powered Robots Run Amok</title>
```

```
    <link href="http://example.org/2003/12/13/atom03"/>
    <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
    <updated>2003-12-13T18:30:02Z</updated>
    <summary>Some text.</summary>
  </entry>
</feed>
```

3.3. Az RSS és az Atom összehasonlítása

Bár alapvetően ugyanazt a célt szolgálják, a két formátum számos helyen eltér. Jól látszik, hogy az Atom későbbi, a felhalmozott tapasztalatokat kihasználó formátum, és jobban figyelembe veszi az érvényben lévő szabványokat. A [9] szerint a következő lényeges különbségek figyelhetők meg:

Tartalom típusa Az RSS 2.0 tartalmazhat sima szöveget vagy szöktetett HTML kódot, de nem mondja meg, hogy valójában melyiket is tartalmazza. Az Atom tartalmazhat sima szöveget, HTML, XML, XHTML és Base64-kódolt tartalmat, vagy hivatkozást egy külső tartalomra, típusmegjelöléssel.

Tartalom Az RSS 2.0 egy `description` elemben tárolja a tartalmat, ami lehet egy összefoglaló vagy teljes szöveg. Az Atom külön `atom:summary` és `atom:content` elemeket definiál, és az `atom:summary` megadása javasolt, ha a tartalom nem szöveges.

Névterek Az Atom névtérben definiált, az RSS 2.0 viszont nem.

Relatív URI-k Az Atom az XML-be beépített `xml:base`-t használja a relatív URI-k feloldásánál. Az RSS 2.0-ban ez nem definiált.

Nyelv Az Atom az XML-be beépített `xml:lang` attribútumot használja a nyelv megjelölésére, az RSS 2.0 saját `language` elemét.

Azonosítók Az Atom megköveteli, hogy minden bejegyzés saját *globálisan egyedi ID-vel* rendelkezzen, ami a bejegyzések megbízható frissítéséhez elengedhetetlen.

Érvényes feed Az Atom 1.0 megengedi, hogy a dokumentum egyetlen bejegyzésből álljon, az RSS 2.0 a teljes feedet fogad el dokumentumként.

Dátumok Az Atom rögzíti a dátumok formátumát az RFC 3339-nek megfelelően (mely az ISO 8601-nek megfelel), az RSS 2.0 nem, ezért ott több különböző formátum terjedt el.

MIME-típus Az Atom MIME-típusa `application/atom+xml`, melyet az IANA regisztrált. Az RSS 2.0 nem rendelkezik regisztrált MIME-típussal, de gyakran az `application/rss+xml`-t használják.

XML séma Az Atom rendelkezik XML-sémával, az RSS 2.0 nem.

Standard Az Atom az IETF által standardizált, nyílt és fejleszthető standard. Az RSS nem standard, és nem fejleszthető szabadon.

Digitális aláírás Az Atom 1.0 rendelkezik arról, hogyan lehet rá XML digitális aláírás alkalmazni.

Elterjedtség Az RSS 1.0 és 2.0 még mindig elterjedtebb és széles körben használt. Általában preferált formátum az Atom-mal szemben. Sok oldal csak egy formátumban publikálja feedjeit, ez gyakrabban az RSS. Gyakran az Atomra is RSS-ként hivatkoznak.

3.4. Konklúzió

A két formátum versengésének a mai napig tanúi lehetünk. A frissített tartalmak és epizodikus weboldalak szállítására számos példát láttunk jóval a technikák elterjedését megelőzően is. A technikák és formátumok időközben folyamatosan fejlődtek, és mára a két komolyabb versenyző maradt, az RSS és az Atom. Amellett, hogy az RSS korábbi és mostanáig is nagyobb ismertséget és népszerűséget élvez, mint az Atom, nem lehet elmenni nyilvánvaló hibái mellett.

Az RSS kezdeti egyszerűségében majdnem használhatatlan volt arra, amire ma használják, Dave Winer ScriptingNews formátuma nagyobb potenciállal rendelkezhetett, ráadásul egy olyan RDF-en alapult, amely végső kiadására megváltozott. A következő verzióban már letisztították az RDF-et, de ezzel inkompatibilissé lett, egyúttal gyakorlatilag magába olvasztotta a ScriptingNews fontos elemeit. A Netscape hanyatlásakor azonban gazdátlanul maradt, és a fejlesztése több ágon folytatódott, ráadásul nem vettek figyelembe számos tényezőt, nem szabályoztak sok kérdést, így aztán a fejlesztők saját belátásuk szerint dolgoztak a formátummal. Több verzióra fel kellett készíteni a szoftvereket (egyes források kilenc különböző, inkompatibilis verziót említenek).

Bár az Atom fejlesztése is megtorpant egy időre, mégis sikerült egy ágon tartani a fejlesztését, amelybe gyakorlatilag bárki beleszólhatott, egy közösség döntött a funkciók, módszerek, formátumok befoglalásáról vagy kizárásáról. A

szervezet végül sikerrel vitte végig az IETF-en a standardizálási folyamatot. Ebben feltétlenül szerepet játszott a projekt nyíltsága és a már meglévő szabványok felhasználása. A fejlesztők számára is egyszerűbb egyetlen verzióval, meglévő eszközökkel és meghatározott szabályok között dolgozni, olyan formátummal, amely ráadásul szabadon fejleszthető.

Fontos megjegyezni, hogy az RSS 1.0 az RDF-et alapul véve jobban átgondoltan, az RDF mint erőforrásleíró nyelv szellemében tartalmazza egy weboldal összefoglalását, amely elősegítheti a szemantikus web fejlődését.[2]

Bár az RSS elerjedtsége jelenleg nagyobb mint az Atomé, munkámat elsősorban az Atom köré építem, egyrészt a magasabb fokú interoperabilitás érdekében, másrészt az Atom mint szabványos formátum terjesztéséért.

[2] A szemantikus web fejlődése nehéz folyamat lehet, mert alapvetően a „tyúk vagy tojás" problémában szenved: amíg nincs fogyasztó, nincs értelme RDF-et előállítani; és fordítva - amíg nincs RDF-et előállító szoftver, nem lesz olyan sem, amely felhasználná azt.

4. fejezet

Atom Publishing Protocol

Az RSS mint formátum már jóval az Atom előtt megjelent, így néhány évnyi tapasztalat összegyűlt a webes megosztás és együttműködés területén, mielőtt megalkották az Atomot. Ebbe a témakörbe tartozik az is, hogyan lehet publikálni egy olyan elemet, amely később egy feedben jelenik majd meg. A blogoknál ez kezdetben egy XML-RPC alapú megoldás volt.

Az Atom nem csupán egy formátum. A készítők szándékosan terveztek hozzá egy protokollt is, az Atom Publishing Protocolt (APP vagy AtomPub), amellyel a szokásos módon manipulálni (elemet hozzáadni, módosítani, törölni) is lehet Atom-feedeket. Az Atom szindikációs formátum és az Atom publikációs protokoll szorosan összetartozik. Az APP az IETF által szabványosított, az RFC 5023 rögzíti. Alapja a HTTP, és az úgynevezett REST alapelveit követi.

4.1. REST

A REST, azaz Representational State Transfer (Reprezentációs állapotátvitel) fogalma a HTTP egyik megalkotójának, Roy Fieldingnek 2000-ben kelt doktori disszertációjában [7] jelent meg először, azóta pedig széles körben használatos a hálózatokkal foglalkozó szakemberek körében. Szigorú értelemben véve a REST alapelvei azt körvonalazzák, hogy bizonyos *erőforrásokat* hogyan lehet definiálni és megfogni, megcímezni. Egy szabadabb értelmezésben egy olyan interfészt írhat le, amely domén-specifikus adat átvitelére alkalmas HTTP-n keresztül[1], további rétegek nélkül (mint például a SOAP, munkamenet-követés és a HTTP coo-

[1] A REST maga nem támaszkodik a HTTP-re, és lehet tervezni olyan egyszerű HTTP-alapú rendszert is, amelyre a REST alapelvei nem teljesülnek, hanem inkább a távoli eljáráshívás (RPC) modellt követi.

kiek). Az olyan rendszereket, amelyek a REST alapelveinek megfelelnek, az angol szakirodalom *RESTful*-nak nevezi.

Egy RESTful alkalmazás a következő feltételek meglétét igényli:

- a funcionalitás erőforrások formájában jelenik meg
- minden erőforrás egyedileg megcímezhető, univerzális módon
- minden erőforrás állapota egy egységes interfészen vihető át a kliens és az erőforrás között, ez az interfész jól definiált műveletekből és tartalomtípusokból áll
- a protokoll kliens-szerver protokoll, mely emlékezetnélküli, cache-elhető és rétegzett.

A hálózati komponensek (kliens és szerver) az erőforrások *reprezentációját* használják arra, hogy manipulálják az erőforrás által képviselt információt, ezt a reprezentációt cserélik ki egymás között egy interfészen. Konkrét példában: az interfész maga a HTTP, az erőforrás azonosítója az URI, reprezentációja pedig valamilyen XML dokumentum. A kommunikáció tetszőleges számú közvetítőn keresztül történhet, például proxy-kon, átjárókon, tűzfalakon keresztül. Az erőforrás-szemlélet lehetővé teszi azt, hogy az alkalmazásnak nem kell feltétlenül tudnia a közvetítő elemekről, elegendő az erőforrás azonosítóját és a szükséges műveletet ismernie.

A REST-et gyakran az RPC-vel (Remote Procedure Call – távoli eljáráshívás) állítják szembe (Fielding ezt eredetileg nem tette). Míg a REST erőforrásokat használ, az RPC parancsokat. Az erőforrások főnevekkel írhatók le, a parancsok igékkel. Az erőforrásokon különböző, egyszerű műveletek végezhetők, de bonyolultabb funkciót már nehezebb megoldani. Az RPC-ben a módszerek összetettsége változó, bonyolultabb megoldásokat is lehet a parancsok mögé rejteni.

A következő példában az RPC-alapú alkalmazás a következő eljárásokat definiálja:

```
getUser()
addUser()
removeUser()
updateUser()
getLocation()
addLocation()
removeLocation()
updateLocation()
listUsers()
listLocations()
```

```
findLocation()
findUser()
```

A kliens kódja pedig nagyjából a következőképp nézhet ki:

```
exampleAppObject = new ExampleApp('example.com:1234');
exampleAppObject.removeUser('001');
```

Hasonló funkcionalitású REST-alapú alkalmazásban erőforrásokat definiálunk:

```
http://example.com/users/
http://example.com/users/user
http://example.com/findUserForm
http://example.com/locations/
http://example.com/locations/location
http://example.com/findLocationForm
```

A fenti erőforrásokban a *user* és a *location* jelöli ki a különböző felhasználók és helyek egyedi azonosítóinak sorát, azaz minden felhasználónak és minden helynek egyedi erőforrása van! Ebben a példában az alkalmazás kódja így nézhet ki:

```
userResource = new Resource('http://example.com/users/001');
userResource.delete();
```

Az elvégezhető műveleteket az interfész definiálja. Szokás szerint ezek általában a létrehozást, módosítást, törlést jelentik. A fenti példa az angol nyelvű Wikipédia REST szócikkéből származik.

4.2. Az APP leírása

Néhány bekezdéssel feljebb volt szó arról, hogy az APP a REST elveit követi, és hogy mennyire így van, azt már a dokumentáció első mondata is jól mutatja. Az RFC 5023 bevezetése így fogalmaz: Az Atom Publishing Protocol egy alkalmazás-szintű protokoll webes erőforrások publikálására és szerkesztésére HTTP és XML 1.0 segítségével. A hangsúly tehát itt is az erőforrásokon van, azok létrehozásán és manipulásán (szerkesztés, törlés) túl pedig lehetőséget ad erőforrás-gyűjtemények kezelésére és felfedezésére. A protokoll nagy mozgásteret ad a szervereknek abban, hogy mit kezdhetnek a kliensek által küldött adatokkal, akár meg is változtathatják azokat, esetleg további műveleteket engedhetnek meg; a kliensszoftvereknek számolniuk kell azzal a lehetőséggel, hogy a szerver

egy az RFC-ben meg sem említett státuszkóddal válaszol, vagy megváltoztatja a küldött adatot. A protokoll különös figyelmet szentel a cache-elhetőségnek.

Az erőforrások HTTP-n keresztül érhetőek el, URI-juk segítségével foghatóak meg. Az APP többféle erőforrást definiál, a legalapvetőbbek a bejegyzés-erőforrások és a média-erőforrások, közös nevükön a *tagerőforrások*. A bejegyzés-erőforrások reprezentációja az Atom szindikációs formátumában megismert *entry* dokumentum. A médiaerőforrások reprezentációja tetszőleges lehet (egy életszerű példában JPEG, PNG vagy más típusú kép- vagy videóállományról lehet szó). A tagerőforrások URI-jai kollekciókban jelenik meg, ezeket szintén erőforrásként lehet elérni, mely URI-val rendelkezik, reprezentációjuk Atom *feed* típusú dokumentum. Ahhoz, hogy a média-erőforrások megjelenhessenek egy kollekcióban, tartozik hozzájuk egy úgynevezett médiacsatoló bejegyzés (media link entry), mely a bejegyzés-erőforrás egy formája. Így a média-erőforrásokhoz mindig tartozik bejegyzés-erőforrás is.

Az APP nem tesz különbséget az Atom feedek és a kollekciókhoz használt feedek között. Az, hogy a kollekció-erőforrások milyen URI-n találhatóak meg, egy *service*-dokumentum írja le. Ez a dokumentum egy szolgáltatást munkaterületekre oszt, amelyekhez kollekciókat kapcsol, megadja azok URI-ját, típusát és a használható kategóriákat (a szindikációs formátum értelmében). A munkaterületek mibenlétét nem definiálja.

Az APP kizárólag a tagerőforrások kezeléséről rendelkezik. Minden művelet HTTP metódusokkal végezhető el. Erőforrás lekérdezése annak URI-jára küldött GET metódussal történik. POST metódus segítségével hozható létre egy új tagerőforrás, melyet a kollekció URI-ján kell végrehajtani. Ha a létrehozandó erőforrás típusa nem bejegyzés, akkor létrejön a médiaerőforrás, és hozzátartozó médiacsatoló bejegyzés is a kollekcióban. A erőforrások elnevezése dinamikus. Egy ismert tagerőforrás módosítása a PUT metódussal lehetséges, a törlésre pedig a DELETE metódus szolgál. A szerver a kérésekre adott válaszban a HTTP-ben meghatározott tetszőleges státuszkóddal válaszolhat, értelmezhet a specifikációban nem meghatározott metódust, és megváltoztathatja a küldött erőforrást, vagy akár megtagadhatja a kiszolgálását. Az ilyen esetek kezelésére a kliensnek kell felkészülnie. Két kérés között megváltozhat az erőforrás tartalma. Ha létrehozás vagy módosítás során megváltozik a küldött tartalom, a szerver célszerűen visszaküldheti a válasz törzsében a megváltozott tartalmat.

4.2.1. Protokoll-dokumentumok

A protokoll két új dokumentumtípust definiál: a kategórialista- és a service-dokumentumot. Természetesen mindkét dokumentumtípus szabályos XML állomány kell, hogy legyen. A fel nem ismert tageket és attribútumokat külső jelölésnek kell tekintenie a feldolgozónak, és nem jelezhet hibát. Az újonan bevezetett elemek egy új névtértben vannak, a `http://www.w3.org/2007/app` névtérben, amelyet rendszerint az `app` előtaggal jelölnek. Az új dokumentumok használják a szindikációs formátumban használt elemeket is, amelyek a `http://www.w3.org/2005/Atom` névtérben szerepelnek, általában `atom` prefixszel. Az itt és az ott használt definíciók megegyeznek.

A kategórialista egy gyűjteményben használható kategóriákat sorol fel. A dokumentum gyökere egy `app:categories` elem, amelynek több `atom:category` elemgyermeke lehet, vagy alternatívaként tartalmazhat egy `href` attribútumot, amely megadja annak a kategórialistának az URI-ját, amellyel ez a dokumentum azonos tartalmat képvisel. Ha vannak felsorolt kategóriák, a `fixed` attribútum jelöli azt, hogy a megadottakon kívül használható-e más kategória (`yes` vagy `no` tartalommal). Megadható egy séma-URI a `scheme` attribútummal, így azok a kategóriák, amelyeknél ez nem adott, ezt a sémát öröklik. Az `atom:category` elemek definíciója megegyezik a szindikációs formátumban megadott definícióval.

A service-dokumentumok arra szolgálnak, hogy felsorolják az elérhető kollekciókat, ezzel biztosítsák a felderíthetőséget. A dokumentum gyökere egy `app:service` elem, amelynek legalább egy `app:workspace` gyermekelemmel rendelkeznie kell. Ezzel az elérhető munkaterületeket sorolja fel. A munkaterületek névvel és kollekciólistával rendelkeznek, ennek megfelelően az `app:workspace` pontosan egy `atom:title` és több `app:collection` gyermekeleme lehet. Az `app:collection` elem `href` attribútuma adja meg az adott kollekció URI-ját. Ezenkívül megadható a kollekció neve, az általa elfogadott dokumentumtípusok listája és a használható kategóriák listája, ezekre rendre az `atom:title`, `app:accept` és `app:categories` elemek szolgálnak, az elsőből pontosan egy, a másik kettőből tetszőleges számban lehet. Az `app:accept` tartalma egy, az RFC 2616-ban meghatározott médiatípus lehet, ami gyakorlatilag egy MIME-típust jelent, hasonlóan a HTTP `Accept` fejlécéhez. Az `app:categories` a korábban említettnek felel meg.

4.2.2. Az APP kiterjesztései a szindikációs formátumhoz

A protokoll számos helyen kiterjeszti a szindikációs formátumot, mivel azt a formátum megengedi. Ennek gyakran az az oka, hogy a protokoll a kollekciókat is feeddel írja le, sőt, a feedeket nem is különbözteti meg. A másik gyakori ok a formátum és a protokoll együttműködésének elősegítése.

- Az `app:collection` elem megjelenhet egy Atom dokumentumban is az `atom:feed` vagy az `atom:source` elem gyermekeként, ahol a kollekciót határozza meg (amellyel szerkeszthető a feed).
- Lehetnek esetek, amikor nem célszerű megjeleníteni a teljes kollekciót. Erre az esetre a szerver egy részleges kollekciót küld a kliensnek. Ekkor a szerver a kollekció (ne feledjük, mint feed) további részeire linkeket helyezhet el. Ez az `atom:feed` gyermekeként megadott `atom:link` elemmel lehetséges, amelynek a `rel` attribútumában kell megadni, hogy az átadott URI milyen kapcsolatban van az aktuális részlettel; ennek értéke `first`, `previous`, `next` vagy `last` lehet, amely így az első, az előző, a következő vagy az utolsó részkollekciót jelöli.
- Az `atom:entry` gyermeke lehet egy `app:edited` elem, amely egy dátum konstrukció, a bejegyzés utolsó szerkesztési időpontját mutatja.
- A protokoll az `atom:entry` elem alatti `link` elem `rel` attribútumában két új opciót vezet be. Ezek az `edit` és az `edit-media`. Az ilyen link jelöli egy kollekciót reprezentáló feedben a tagerőforrás URI-ját, amelyen keresztül az erőforrás módosítható vagy törölhető. A bejegyzés-erőforráshoz tartozik az `edit` értékű link. Ha a kollekció médiát sorol fel, akkor az `edit-media` által mutatott URI tartozik a média-erőforráshoz, a másik annak médiacsatoló bejegyzéséhez (mint bejegyzés-erőforráshoz).
- Az `atom:entry` elem alatt bevezetésre kerül az `app:control`. Ez tartalmaz egy további `app:draft` elemet, amelynek értéke `yes` vagy `no` lehet. Ez jelöli azt, hogy a létrehozandó bejegyzés vázlat állapotban van, vagy már publikálható a külvilág felé.

4.2.3. Az APP kiterjesztései és megjegyzései a HTTP-hez

Az előírt funkcionalitás eléréséhez szükséges a HTTP kis mértékű kiterjesztése is. Ez főleg akkor válhat szükségessé, ha a protokolldokumentumok nem fejezik ki megfelelően az eredményt, vagy használatuk nem célszerű.

– Nem a HTTP kiterjesztése, de szorosan összefügg azzal: az Atom dokumentumok érvényesek, akár feed, akár entry típusúak. Mindkét dokumentumtípust az `application/atom+xml` MIME-típus jelöli, ez bizonyos esetekben nem elegendő. Ezért a protokoll megengedi ezután a `type` paraméter használatát, értéke `entry` vagy `feed` lehet. Így tehát egy entry típusú Atom dokumentum MIME-típusa: `application/atom+xml; type=entry`.

– A bejegyzések elnevezése teljesen dinamikus, a szerver dönti el a nevet és az elérés URI-ját létrehozáskor. Azonban a kliens is javasolhat egy nevet, a POST metódus hívásakor a fejlécben átadható a `Slug` paraméter, ez a javasolt nevet jelöli. A szerver ezt megváltoztathatja, átalakíthatja, vagy akár figyelmen kívül is hagyhatja.

– Létrehozásnál a szerver válasza siker esetén `201 Created` lesz. A törzsben visszaküldött dokumentum azonban nem feltétlenül a teljes dokumentum, lehet annak egy része is. Az erőforrás URI-ját a fejléc `Location` része tartalmazza.

– Média-erőforrás létrehozásánál egyből két erőforrás is létrejön, egy a médiának, egy pedig a médiacsatoló bejegyzésnek. Ez a két különböző erőforrás a `Location` és a `Content-Location` fejlécekkel fejezhetők ki egyidőben.

– Az authentikáció tekintetében a protokoll nagyon rugalmas, és azt mondja, hogy bármilyen séma alkalmazható az illetéktelen hozzáférések ellen, de a minél nagyobb kompatibilitás érdekében egy Basic authentikáció implementálását javasolja TLS (Transport Layer Security - Transzport rétegbeli védelem) felett. Ha egy authentikációs kísérlet sikertelen, a szerver státuszkódja `401 Unauthorized` vagy `403 Forbidden` lehet.

– A szerver megtagadhatja például egy új bejegyzés felvételét akkor, ha az ahhoz megadott kategória nem szerepel a kollekció kategórialistájában, amennyiben az rögített. Szintén megtagadhatja a szerver akkor, ha a kollekció nem támogatja az átadott típusú dokumentumot. (Például az `app:collection`-jei között nem található olyan `accept`, amelyet a kliens át akar adni.)

– A kiszolgálónak és a kliensnek figyelmet kellene szentelnie a cache-elhetőségre. Ez úgy érhető el, hogy amikor a szerver egy erőforrást ad vissza a reprezentációjában, a fejlécek között lehet egy `ETag`, amely az erőforrás kivonatát jelentheti. A kliens ekkor lekérheti az erőforrást a kérés fejlécében

az `If-None-Match` paraméterben a korábban ismert kivonattal, és ha az nem változott, a szerver a `304 Not Modified` státusszal válaszolhat. Ugyanígy, PUT metódussal történő módosítás esetén használható a `If-Match` fejléc, és ha a szerver ezt figyelembe veszi, lehetséges, hogy nem hajtja végre a módosítást, ha az erőforrás tartalma időközben megváltozott (szokásos státuszkód: `412 Precondition Failed`).

4.3. Példák

A következőkben bemutatok néhány lehetséges módot, kérésekkel és a szerver válaszaival. A protokoll legáltalánosabb felhasználási módja a blogok írása; ehhez minden adott: a felhasznált képek, animációk és minden más egyszerűen kezelhetők média-erőforrásként, míg magukat a tényleges bejegyzéseket a bejegyzés-erőforrások reprezentálják. A blogbejegyzések ezután megjelenhetnek egy weboldalba ágyazva, egyedi URL-en olvashatók. A [13]-ból származó példák következnek.

Mielőtt megkezdődhet a feedek kezelése, fel kell deríteni azokat. Ez a service-dokumentum URI-jára küldött GET kéréssel történhet. Az alábbi példában látható service-dokumentum két munkaterületet definiál. Az első munkaterület egy blogot határoz meg két kollekcióval, az egyik a bejegyzéseknek, a másik a bejegyzésekben használt képek számára. A bejegyzésekben használható kategóriákat egy külső kategórialista határozza meg, de attól el is lehet térni. A képek számára fenntartott kollekció a megadott típusú képeket fogadja el. A másik munkaterület egy mellékblogot ír le, az ebbe küldhető bejegyzések kategóriája kötött listából lehet.

```
<?xml version="1.0" encoding='utf-8'?>
<service xmlns="http://www.w3.org/2007/app"
         xmlns:atom="http://www.w3.org/2005/Atom">
  <workspace>
    <atom:title>Main Site</atom:title>
    <collection href="http://example.org/blog/main" >
      <atom:title>My Blog Entries</atom:title>
      <categories href="http://example.com/cats/forMain.cats" />
    </collection>
    <collection href="http://example.org/blog/pic">
      <atom:title>Pictures</atom:title>
      <accept>image/png</accept>
      <accept>image/jpeg</accept>
      <accept>image/gif</accept>
    </collection>
```

```
    </workspace>
    <workspace>
      <atom:title>Sidebar Blog</atom:title>
      <collection href="http://example.org/sidebar/list" >
        <atom:title>Remaindered Links</atom:title>
        <accept>application/atom+xml;type=entry</accept>
        <categories fixed="yes">
          <atom:category scheme="http://example.org/extra-cats/" term="joke" />
          <atom:category scheme="http://example.org/extra-cats/" term="serious" />
        </categories>
      </collection>
    </workspace>
  </service>
```

Itt egy bejegyzés létrehozása látható. A példa mutatja azt is, hogy a létrehozáshoz a kliens Basic authentikációt használ. A Slug fejléc javasolja az új bejegyzés URI-ját.

```
POST /edit/ HTTP/1.1
Host: example.org
User-Agent: Thingio/1.0
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
Slug: First Post

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
  <author><name>John Doe</name></author>
  <content>Some text.</content>
</entry>
```

A szerver válaszában megtalálható az elem edit-URI-ja és kivonata (ETag). Ezenkívül a bejegyzés is megváltozik: kiegészül az edit-URI-val, amely a Slug figyelembevételével jön létre.

```
HTTP/1.1 201 Created
Date: Fri, 7 Oct 2005 17:17:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
Location: http://example.org/edit/first-post.atom
ETag: "e180ee84f0671b1"

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2003-12-13T18:30:02Z</updated>
```

```
    <author><name>John Doe</name></author>
    <content>Some text.</content>
    <link rel="edit" href="http://example.org/edit/first-post.atom"/>
  </entry>
```

A feedelem lekérdezése `If-None-Match` fejléccel. A szerver válasza: `304 Not Modified`.

```
GET /edit/first-post.atom HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
If-None-Match: "e180ee84f0671b1"
```

A következő példa egy módosítási kísérlet, az If-Match fejléc használatával. Ha a szerver ezt figyelembe veszi és a kivonat egyezik, a válasza `200 Ok` lesz, ha a kivonat nem egyezik, `412 Precondition Failed` vagy `409 Conflict` lehet (a szerver implementációjától függően).

```
PUT /edit/first-post.atom HTTP/1.1
Host: example.org
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Type: application/atom+xml;type=entry
Content-Length: nnn
If-Match: "e180ee84f0671b1"

<?xml version="1.0" ?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>Atom-Powered Robots Run Amok</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2007-02-24T16:34:06Z</updated>
  <author><name>Captain Lansing</name></author>
  <content>Update: it's a hoax!</content>
</entry>
```

A bloghoz most hozzáadunk egy képet, ezt az eredeti bináris reprezentációjában tesszük.

```
POST /edit/ HTTP/1.1
Host: media.example.org
Content-Type: image/png
Slug: The Beach
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

bináris adat...
```

A kiszolgáló válasza már a képhez tartozó médiacsatoló bejegyzést tartalmazza. Ez gyakorlatilag a metaadatokat tárolja. Létrehozása után mind a média, mind a metaadat módosítható.

```
HTTP/1.1 201 Created
Date: Fri, 7 Oct 2005 17:17:11 GMT
Content-Length: nnn
Content-Type: application/atom+xml;type=entry;charset="utf-8"
Location: http://example.org/media/edit/the_beach.atom

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>The Beach</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6a</id>
  <updated>2005-10-07T17:17:08Z</updated>
  <author><name>Daffy</name></author>
  <summary type="text" />
  <content type="image/png" src="http://media.example.org/the_beach.png"/>
  <link rel="edit-media" href="http://media.example.org/edit/the_beach.png" />
  <link rel="edit" href="http://example.org/media/edit/the_beach.atom" />
</entry>
```

Ha minden szükséges képfile felkerült a szerverre, és tudjuk azok URL-jeit, elkészíthetjük a bejegyzést.

```
POST /blog/ HTTP/1.1
Host: example.org
Content-Type: application/atom+xml;type=entry
Slug: A day at the beach
Authorization: Basic ZGFmZnk6c2VjZXJldA==
Content-Length: nnn

<?xml version="1.0"?>
<entry xmlns="http://www.w3.org/2005/Atom">
  <title>A fun day at the beach</title>
  <id>urn:uuid:1225c695-cfb8-4ebb-aaaa-80da344efa6b</id>
  <updated>2005-10-07T17:40:02Z</updated>
  <author><name>Daffy</name></author>
  <content type="xhtml">
      <xhtml:div xmlns:xhtml="http://www.w3.org/1999/xhtml">
          <xhtml:p>We had a good day at the beach.
              <xhtml:img alt="the beach"
                  src="http://media.example.org/the_beach.png"/>
          </xhtml:p>
          <xhtml:p>Later we walked down to the pier.
              <xhtml:img  alt="the pier"
                  src="http://media.example.org/the_pier.png"/>
          </xhtml:p>
      </xhtml:div>
  </content>
</entry>
```

4.4. GData API

A Google is kiterjedten használja az Atom publikációs protokollt szolgáltatásainak eléséhez [11]. Atom-on keresztül lehet elérni a Gmail levelezésünket, a Calendar bejegyzéseit, vagy akár táblázatainkat a Docs szolgáltatásban; majd' mindegyik szolgáltatása elérhető feedeken keresztül, RSS, Atom vagy sok esetben JSON (JavaScript Object Notation) formátumban. Az RSS és az Atom kiterjesztési lehetőségeit erőteljesen kihasználja a szabvány adta lehetőségeken belül. Maga a GData protokoll feed létrehozására nem ad lehetőséget, de az egyes szolgáltatások ezt megtehetik. A GData az APP lehetőségein kívül különböző lekérdezéseket és optimista konkurrenciakezelést biztosít.

A lekérdezések biztosítása azt jelenti, hogy egy feedben lévő elemek szűrhetőek a feed lekérésekor GET-paraméterek segítségével. Alapesetben a teljes feed megjelenik lekérdezéskor. A `q` paraméter megadásával a feedelemeken teljesszöveges (full-text) keresést hajthatunk végre, vagyis csak azok az elemek jelennek meg a feedben, amelyekre a megadott szöveg illeszkedik. A `category` paraméter használható a kategóriák szerinti szűrésre, a paraméterben használható a `|` (függőleges vonal) jel a *vagy*, illetve a `,` (vessző) az *és* operátorok jelölésére. Az `author` paraméter teszi lehetővé az elemek szűrését azok szerzője szerint. Az `updated-min`, `updated-max`, `published-min`, `published-max` értelemszerűen az elem frissítésére illetve létrejöttére fogalmaznak meg feltételt, értékük az RFC 3339 szerinti időbélyeg lehet. A lekérdezés kimenetét befolyásolja néhány további paraméter, ezek: `start-index` (1-től, az első megjelenítendő elem indexe), `max-results` (az elemek maximális száma), `alt` (alternatív reprezentáció, lehetséges értékei: `atom`, `rss`, `json`, `json-in-script`; alapértelmezett érték: `atom`).

További két lehetőség van szűrésre. Egyrészt, a feed URL-jéhez a */elemId* hozzáfűzésével létrejött URL-en mindig az adott azonosítójú feedelemet lehet lekérdezni. Másrészt, a feed URL-jét kiegészítve a */-/kategória1[/kategória2]...* sorozattal, a kapott URL-en a megadott kategóriák szerinti feedelemek fognak szerepelni.

A lekérdezésben kapott feedekben további elemek jelennek meg. A `feed` elem gyermekeként megtalálható lesz az `openSearch:totalResults`, `openSearch:startIndex` és `openSearch:itemsPerPage`, ezek tartalma rendre a feedelemek teljes száma, az ebben a lekérdezésben megjelenő első feedelem indexe és az ebben a lekérdezésben megjelent feedelemek száma. Az oldalak közötti lapozást az APP szabályozásának megfelelően a `feed` elem alatt található azon `link` elemekkel le-

het megoldani, amelyekben a `rel` attribútum értéke `previous` vagy `next`.

Az optimista konkurrenciakezelés megvalósítása hihetetlenül egyszerű és véleményem szerint zseniális. Egy adott feedelem URL-je mindig a feed URL-jéhez konkatenált perjel, majd az entry azonosítója. Ennek a feedelemnek az edit-URI-ja az ehhez az URL-hez fűzött perjel, és a feedelem verziószáma, így, ha szerkesztjük, ez az URI azonnal érvénytelenné válik, a későbbi lekérdezésekben már az új edit-URI jelenik meg. Ez lehetetlenné teszi, hogy amíg egy kliens módosította egy feedelem tartalmát, a másik ezután azt felülírja vagy letörölje a korábbi verzió alapján.

4.4.1. Authentikáció

A legtöbb esetben a kliensalkalmazások egy bizonyos felhasználó nevében működnek. Ehhez a felhasználónak meg kell adnia nevét és jelszavát. A GData API többféle authentikációs módot is támogat. Az önálló asztali alkalmazások számára a ClientAuth módszert ajánlja; ha a kliens egy többfelhasználós webes alkalmazás, akkor célszerűbb az AuthSub vagy az OAuth módot választani. Az iGoogle- vagy OpenSocial-alapú alkalmazásokhoz is létezik módszer.

AuthSub

Ez a módszer a webes alkalmazásokhoz javasolt. A nagyobb biztonság érdekében az AuthSub interfész átveszi az alkalmazástól a felhasználó beléptetésének feladatát, azaz annak nem kell a felhasználó jelszavát kezelnie. A működése a következő: webalkalmazás (weboldal) átirányítja a felhasználót a `https://www.google.com/accounts/AuthSubRequest` oldalra, úgy, hogy közben annak átad egy GET-paramétert, a `next`-et: ez tartalmazza annak az oldalnak az URL-jét, amelyre a felhasználó visszakerül a weboldalon. Az AuthSub rendszer bekéri a felhasználó adatait, s ha ez sikerül, a böngészőt visszairányítja a `next` paraméterben átadott oldalra, és annak egy `token` nevű GET-paraméterben visszaad egy tokent, amely a továbbiakban azonosításra használható. Ennek módja a következő: a kliens az authentikációt igénylő HTTP-kérés fejlécei között az `Authorization` fejlécen az `AuthSub token="`*`korabbanKiadottToken`*`"` értéket adja át. Ez egészen hasonló az AtomPub által javasolt megoldáshoz.

Ahhoz, hogy az adathoz végül valóban hozzáférjünk, a token kérésénél a `scope` paraméterben átadott értékkel kell jelezni, melyik Google-szolgáltatáshoz szeretnénk hozzáférni.

Az AuthSub további lehetőségei:

- az alkalmazás egy regisztrációt követően használhat biztonságos bejelentkeztetést. Ha erre szükség van, a token kérésénél a `secure` paraméterben `1` értéket kell átadni
- ha az alkalmazás a felhasználó adatait nem egy lekérésre, hanem egy teljes munkamenetre (session) kéri, ezt a `session` paraméter `1` értékével kell jelezni
- használható a Google Apps szolgáltatásában regisztrált doménekben is, ehhez a `hd` paraméterben a kívánt domén nevét kell átadni

OAuth

Az OAuth egy széles körben elfogadott API az authentikációs eljárások terén. Sok tekintetben hasonlít az AuthSub módszerre abban az esetben, amikor azt regisztrációval és fokozott biztonsággal használjuk. Az OAuth használatához is regisztrálni kell az alkalmazást a Google-nél, majd egy biztonsági tanusítványt kell feltölteni, ez alapfeltétel ahhoz, hogy használni lehessen. Egy token kerül kiosztásra, három lépésben:

- a webalkalmazás egy request-tokent igényel a `https://www.google.com/accounts/OAuthGetRequestToken` végponton, megadva GET- vagy POST-paraméterként az OAuth-hoz szükséges adatokat (verziószám, nonce, timestamp, `consumer_key`, `signature_method`) valamint a már ismertetett paramétert, amely behatárolja a használni kívánt adattárat. A válaszban megkapja a tokent és a hozzátartozó titkos kulcsot
- a webalkalmazás authorizálást kér az igényelt tokenre, a `http://www.google.com/accounts/OAuthAuthorizeToken` végponton, az `oauth_callback` paraméter tartalmazza azt az URL-t, amelyre sikeres azonosítás esetén átirányítja a böngészőt a Google oldala. Természetesen a tokent is meg kell adni a paraméterek között
- a webalkalmazás hozzáférési tokenre cseréli a request-tokent, a `https://www.google.com/accounts/OAuthGetAccessToken` URL-en POST metódussal, átadva a korábbi request-token adatait. A válasz tartalmazza az új hozzáférési tokent és a hozzátartozó titkos kulcsot

Mint látható, minden kérés HTTPS-en zajlik, ennek titkosításához a feltöltött tanusítványt kell használni. Ezután a kérést a megfelelő feed-en az `Authorization`

fejléccel, `OAuth` sémában kell intézni a szerver felé, megadva az új tokent és a hozzá kapcsolódó adatokat.

ClientLogin

Ez a legegyszerűbb módja az authentikációnak a Google szolgáltatásaiban, ezt azonban az asztali alkalmazásokhoz ajánlja a Google. Ebben az esetben az alkalmazás maga gyűjti be vagy tárolja a felhasználó nevét (emailcím) és jelszavát, és a `https://www.google.com/accounts/ClientLogin` URL-re POST-kérést intéz, átadva paraméterként ezeket és az elérni kívánt szolgáltatás nevét. A HTTP-válasz tartalmazza a tokent, amelyet a szokásos módon, az `Authorization` fejléc megadásával, a `GoogleLogin` séma használatával lehet átadni a kívánt feed URL-jére küldött kérésben.

4.4.2. Szolgáltatások

A Google különböző szolgáltatásai nyilvánvalóan különböző jellegű adatokhoz adnak hozzáférést, különféle módokon. Sajnos itt nincs lehetőség arra, hogy teljes részletességgel bemutatásra kerüljenek a különböző típusú adatok elérésének módjai, ám a szolgáltatásokat néhány mondatban összefoglalom.

Minden szolgáltatásban közös az általuk használt szótár. Az adatelemek természetesen feedelemekként (`<entry>`) jelennek meg, amelyben a rendszerint `gd` prefixszel jelölt `http://schemas.google.com/g/2005` névtérbe tartozó XML-elemek írják le, adják meg az adott típusú (a Google terminológiájában: *kind*) adatelem megfelelő értékeit. Ha egy feed egy adott típushoz tartozó adatelemeket kezel, akkor ezt az `atom:category` XML-elemmel adja meg, ahol a kategória sémája (`scheme` attribútum) `http://schemas.google.com/g/2005#kind`, maga a kategória (`term` attribútum) pedig `http://schemas.google.com/g/2005#`*`tipusNeve`*.

A Google által kezelt legfontosabb típusok külön is megadottak, ezek a Contact, Event illetve a Message típusok. Például a Contact típusban a feedelemben a kötelező XML-elemek mellett megjelenik a `gd:email`, `gd:im`, `gd:phoneNumber`, amelyek rendre a kontakt emailcímét, azonnaliüzenetküldő-címét (instant messenger, IM) illetve telefonszámát írják le. A kapcsolat neve az `atom:title` elem tartalma. Az Event típusban a `gd:when`, `gd:here`, `gd:who` elemek játszanak fontosabb szerepet. A Message típus elemeiben a `gd:who` elemek játszanak szerepet. Ez utóbbinál a feedelem kategóriája függ attól, hogy email, azonnali üzenet, kimenő üzenet vagy spam az adott üzenet közelebbi meghatározása, ennek megfelelően

lehet `http://schemas.google.com/g/2005#message.inbox`, `#message.sent`, `#message.chat`, illetve `#message.spam`. A `gd:who` elem `rel` attribútuma határozza meg, hogy a tartalmában megadott személy a küldő, címzett, vagy például titkos másolat címzettje.

Google Base

Ez a szolgáltatás lehetővé teszi különböző cikkek, tételek leírását, amelyek pedig később kereshetővé válnak mások számára. Jelenleg csak az USA-ban és Németországban üzemel. A keresésre közzétehető cikkek között a legnépszerűbb fajták: események, hotelek, ingatlanok, munkalehetőségek, termékek, tanfolyamok, vizsgálati cikkek (review-k), receptek, járművek, bérlemények, és rengeteg további típusú adat tehető kereshetővé. Az adatelemekhez attribútumokat kapcsol a rendszer, amelyeket `gd` névterű XML-elemek írnak le.

Blogger

A Google által felvásárolt szolgáltatás. Mivel az APP gyakorlatilag a blogok kezeléséhez ideális, a GData alig tesz hozzá új elemeket, leszámítva a lekérdezéseknél tárgyalt dolgokat.

Google Calendar

A Google naptárszolgáltatása, amely egy felhasználónak több naptár kezelését és a naptárak felhasználók közötti megosztását is lehetővé teszi. Ebben az esetben létezik a naptáraknak egy feedje és a naptárakon belül az eseményeknek egy-egy feedje. A naptárspecifikus adatok leírására, például az időzóna, az esemény rejtett státusza, a naptárban megjelenő szín megadására a `http://schemas.google.com/gCal/2005` névtér ad lehetőséget, rendszerint `gCal` prefixszel jelölten.

Google Code Search

A nyílt forráskódú projektek lehetővé teszik forráskódjuk kereshetőségét, melyet a Google a Code Search szolgáltatásában használ ki. A keresés nem csak a webes felületen böngészőből indítható, feedet is biztosít hozzá az óriásvállalat. Ehhez mindössze egy GET-kérést kell intézni a `http://www.google.com/codesearch/feeds/search` URL-re a szokásos `q` paraméterrel, a válasz pedig feedben érkezik.

A `http://schemas.google.com/codesearch/2006` névtér szokásos rövidítése `gcs`, melyben a `gcs:file` jelöli a file nevét, amelyben a találat van, a `gcs:match` adja meg a találat szövegét és hogy ez hanyadik sorban van a file-ban, valamit `gcs:package` egy URI-t biztosít a file-ra (verziókezelő szerver címe vagy egy tömörített állomány URL-je).

Google Contacts

Ez a Google-szolgáltatások által közösen elért adat, a felhasználó által ismert kapcsolatokat tárolja. A feedelemek típusa a korábban említett Contact.

Google Book Search

Hasonlóképp a Code Search-höz, a Book Search az elérhető könyvek tartalmában végez teljesszöveges keresést, és az eredményeket feed formájában bocsátja a felhasználó rendelkezésére.

Google Documents List

A Google Documents online, böngészőben szerkeszthető dokumentumokat tárol, ezekhez biztosít hozzáférést egy feeden keresztül. Ilyen módon a dokumentumok nem szerkeszthetők, ám a tartalmuk lecserélhető a médiaerőforráson keresztül (`<link rel="edit-media" ... />`).

Google Finance

Ez a szolgáltatás egyrészt valósidejű (vagy késleltetett) tőzsdei adatokat biztosít, másrészt lehetővé teszi saját portfoliók nyilvántartását. A pénzügy-specifikus adatok leírását a `http://schemas.google.com/finance/2007` névtér elemei biztosítják.

Google Health

A Google Health azért jött létre, hogy tárolja és kezelje felhasználóinak egészségügyi adatait, információt, illetve betegségek, rendellenességek esetén segítséget nyújtson bizonyos hasznos információkkal. Feljegyzések vihetőek fel tünetekről, importálhatók korházi adatok és figyelemmel kísérhető egy egész kórtörténet; a rendszer a beavatkozások egymásrahatásáról is ad információkat. Valamint, beállíthatók gyógyszerek szedésének rendje. A Google feedjeiben a feed-

elemeket a Continuity of Care Record (CCS) szabvány XML-formátumából vett elemekkel egészíti ki.

Google Notebook

A Notebook szolgáltatás jegyzetek készítéséhez ad segítséget az interneten, valamint könyvjelzőket tárol. (Ennek a műnek egyes részei is ezen szolgáltatás segítségével készültek.) A szolgáltatásban egyaránt készíthetők magánjellegű és publikus jegyzetek. A GData API csupán a nyilvános jegyzetek feeden keresztüli olvasására ad lehetőséget. Ehhez két feed tartozik, egyik a felhasználó elérhető jegyzettömbjeit listázza, a másik pedig egy kiválasztott jegyzettömb jegyzeteit. A feedek nem igényelnek authentikációt és nem írhatóak, de a GData többi tulajdonsága, különösen a lekérdezhetőség érvényes marad.

Picasa Web Albums

A Picasa Web Albums lehetővé teszi fényképek elhelyezését, ezáltal azok ismerősökkel való megosztását. Publikus és privát albumok létrehozását, fényképek geotagelését (geográfiai adatokkal való felcímkézését), jegyzetek írását is támogatja. Ennek megfelelően a Picasa feedek igen sok külső szótár importálását igénylik. Az API lehetővé teszi az albumok teljeskörű kezelését feeden keresztül, minden adatával együtt (a különböző névterekben lévő elemek segítségével). Egy album `self` értékű rel-linkje egyben az adott album feedje is, amelyen keresztül az albumhoz tartozó képek kezelhetők.

Ez a feed tipikusan médiafeed, vagyis a kép eredeti bináris reprezentációjában kezelhető a GData API-val (mint az APP egy implementációjával). Érdekes megoldás azonban, hogy ez a feed együtt kezeli a médiacsatoló-bejegyzéseket és média-erőforrásokat. Az egyes feedelemek `edit` és `edit-media` értékű `rel` attribútummal rendelkező linkjei általában ugyanarra az URI-ra hivatkoznak. Ezen URI-n szerkeszthető a metaadathalmaz és a kép tartalma is, sőt, akár együtt is, a következőképp: a feed elfogadja a `Content-Type: multipart/related` fejlécű feltöltéseket is, amelyben az egyik résztartalom MIME-típusa `application/atom+xml`, a másiké pedig egy elfogadott képformátum típusa vagy egy videoformátum típusa (például `image/jpeg` vagy `video/mpeg4`).

További feedek léteznek az egyes képekhez rendelt címkék felsorolásához, címkék hozzárendeléséhez, megjegyzések kezeléséhez, valamint az időbeli változások követéséhez.

Google Spreadsheets

A Google Spreadsheets korábban különálló szolgáltatásként működött, mostanra beolvadt a Google Documents-be, és egységes felületen használatos. Ám azok között egyedülálló módon lehetőséget ad a táblázatok kezelésére, feedeken keresztül. A szolgáltatás elsődleges feedje megadja a munkafüzetek listáját, illetve azok feedjeire mutató linkeket. Ezen feedeken kilistázhatók az adott munkafüzetek tartalma, munkalapok listájában. A munkalapok listájából kiválaszott egyik munkalap feedje kétféle módon ábrázolja a munkalapot: sorok listájában illetve cellák listájában. Ezeken keresztül a munkafüzetek, táblázatok teljes mértékben szerkeszthetők.

Google Webmaster Tools

A Webmaster Tools webmesterek számára kínál egy kiváló lehetőséget: megmutatja, hogyan látja a Google keresőmotorja ide beregisztrált oldalait. Ez azt jelenti, hogy diagnosztizálja tartalmát és jelenti, ha problémát talál, megmutatja, mik a leggyakoribb keresőszavak, amelyekre megtalálják az oldalt, hogy mikor volt utoljára frissítve a Google adattárházában, az oldalon található linkekről készít statisztikákat, és számos más adatot juttat a weboldal tulajdonosának tudomására. A feedek minden beállítást elérhetővé tesznek a GData API-n keresztül.

YouTube

A YouTube-ot talán senkinek nem kell bemutatni: ez a világ legnagyobb hálózati forgalmat generáló oldala – egy videómegosztó oldal, amelyre percenként 13 órányi videóanyagot töltenek fel. Miután a Google felvásárolta a YouTube-ot, azonnal hozzálátott rendszerének beintegrálásába, így a Google felhasználói is saját azonosítójukkal használhatják az oldalt. A YouTube videók is elérhetők, kereshetők, szerkeszthetők Atom-feedeken keresztül. Ez az API hihetetlenül összetett, bonyolult rendszert vetít a külvilág felé.

5. fejezet

Atomizer: Atom-feed weboldalból

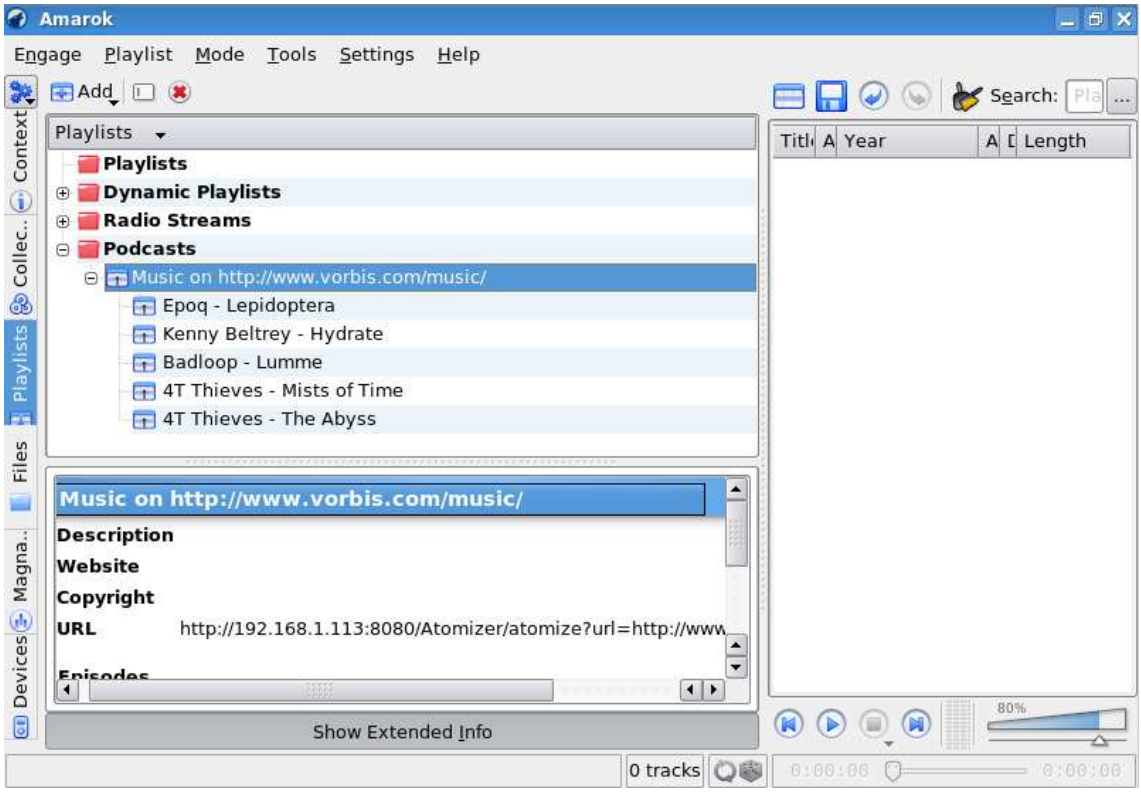
5.1. Motiváció

Célom egy nagyon egyszerű alkalmazás elkészítése volt, amely jól példázza a feedek sokoldalú hasznosíthatóságát. A végeredmény egy olyan webalkalmazás lett, amely egy weboldalon linkelt zeneszámokat egy olyan listába gyűjti, amelyet egy alkalmas lejátszó alkalmazás megért, és lejátszási listaként használhatja, illetve rendszeresen frissítheti ezt a listát, az eredeti weboldal frissítésének megfelelően. A webalkalmazásnak az *Atomizer* nevet adtam, mert az Atom formátumot használja fel a lista szállításához. A végeredmény egy úgynevezett podcast.

5.2. Felhasznált elemek és környezet

Az alkalmazás elkészítéséhez a Java nyelvet választottam, egyrészt, hogy megismerkedjek a servletekel, másrészt a feedeket kezelő API-k sokasága miatt. Ezek közül én a ROME rendszert választottam: ezt találtam a legkiforrottabb keretrendszernek. Nem szabad azonban megfeledkezni a többiről sem, a legfontosabbak: Groovy, Atom4J, Jakarta FeedParser.

A munkám során kizárólag nyílt forráskódú szoftvereket használtam fel. Így például a servletkonténerként az Apache Tomcatet, fejlesztői környezetként a Eclipse rendszert használtam. Az alkalmazás elkészítéséhez szükségem volt még egy komponensre, a JTidy-ra: ez képes a szabálytalan HTML-oldalakat szabályos formára hozni és végül elemezni; szerencsére ez is nyílt forrású. A ROME függőségeként a JDOM nevű rendszer is szükséges volt a fejlesztéshez.



5.1. ábra. Amarok: a feed értelmezve

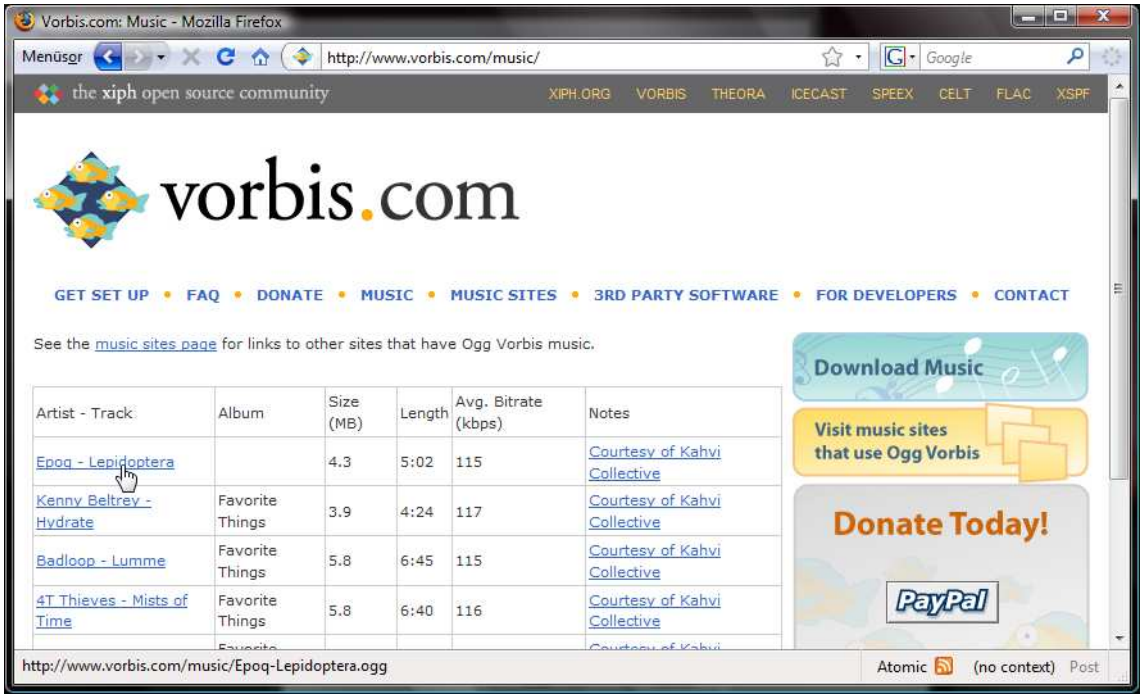
5.2.1. Apache Tomcat

A Tomcat egy jól ismert servletkonténer, az Apache Software Foundation gondozásában: egyszerű, tiszta Java-alapú webszerverkörnyezetet biztosít Java kódok futtatásához. Én az 5.5.25-ös verziót használtam, amely a Java Enterprise Edition Servlet 2.4 és JSP 2.0 verzióját támogatja. A Tomcat nyílt forráskódú szoftver, így szabadon felhasználható bármilyen célra, teljesen kézenfekvő választás egy kisméretű webalkalmazás elkészítéséhez.

A munkám során messze nem használtam ki a Tomcat összes lehetőségét, JSP-t is csak érintőlegesen.

5.2.2. ROME

A ROME egy osztálykönyvtár, amely Atom és RSS feedek beolvasására, reprezentálására és generálására szolgál. Támogatja az RSS és Atom formátumok minden változatát, céljai közé is tartozik, hogy egyetlen interfész segítségével kezeljen minden ilyen feedet. Jelenleg a 0.9-es verzió áll rendelkezésre. Nyílt forráskódú projekt, az Apache Licence 2.0 alatt adják ki. Mindössze egyetlen osztály-



5.2. ábra. A Vorbis audiokodek zenei oldala

gyűjteményre, a JDOM-ra támaszkodik [12].

Bár a ROME képes egységesen kezelni a feedet, annak tulajdonságait, a feed-elemeket és azok tulajdonságait is, függetlenül a végül kiválasztott formátumtól, én első körben a Atom formátumra koncentráltam, és közvetlenül az Atomnak megfelelő formátumban hoztam létre a feedet osztályszinten. Fejlesztési lehetőség, hogy az Atomizer ne csak Atom formátumban szolgáltasson feedet, mert bár meggyőződésem, hogy az RSS formátuma „nehézkes” és nem szabványos, nem szabad elfelejteni, hogy még mindig elterjedtebb. Sajnos találkoztam olyan aggregátorral, amely nem volt képes értelmezni az előállított Atom-feedet. RSS-t minden bizonnyal sikerült volna.

5.2.3. JDOM

A JDOM osztálygyűjtemény XML állományok belső reprezentálására almalmas. A World Wide Web Consortium (W3C) DOM specifikációjához hasonló, azonban a Java nyelv és környezet sajátosságait figyelembe vevő, jóval egyszerűbben kezelhető rendszer, amely XML állományok betöltésére és szerializálására nem képes, de gyakorlatilag bármely ilyen implementációval együttműködik, így a Java alapértelmezett SAX és DOM megvalósításával is. Támogatja az

XPath-t és az XSLT-t is. Ez is nyílt forráskódú szoftver, licensze az Apache Licencehez hasonló, de annál megengedőbb.

A JDOM nem játszott közvetlen szerepet a fejlesztésben, csak a ROME függőségeként került be.

5.2.4. JTidy

A JTidy a HTMLTidy Java nyelvű portja, ennek megfelelően képes HTML forráskódot ellenőrizni, javítani, és beolvasni – sőt, DOM interfészt biztosít a beolvasott tartalom feldolgozására. Ez az egyik alapját képezi az alkalmazásnak: ez segít begyűjteni a weboldalon található hiperhivatkozásokat, oly módon, hogy a HTML-kód értelmezése során rendezi a kódot, majd XML-ként értelmezve azt DOM-fát épít. Ebben a fában XPath segítségével kereshetővé válnak a weboldalon elhelyezett linkek.

5.2.5. Fejlesztői környezet: Eclipse

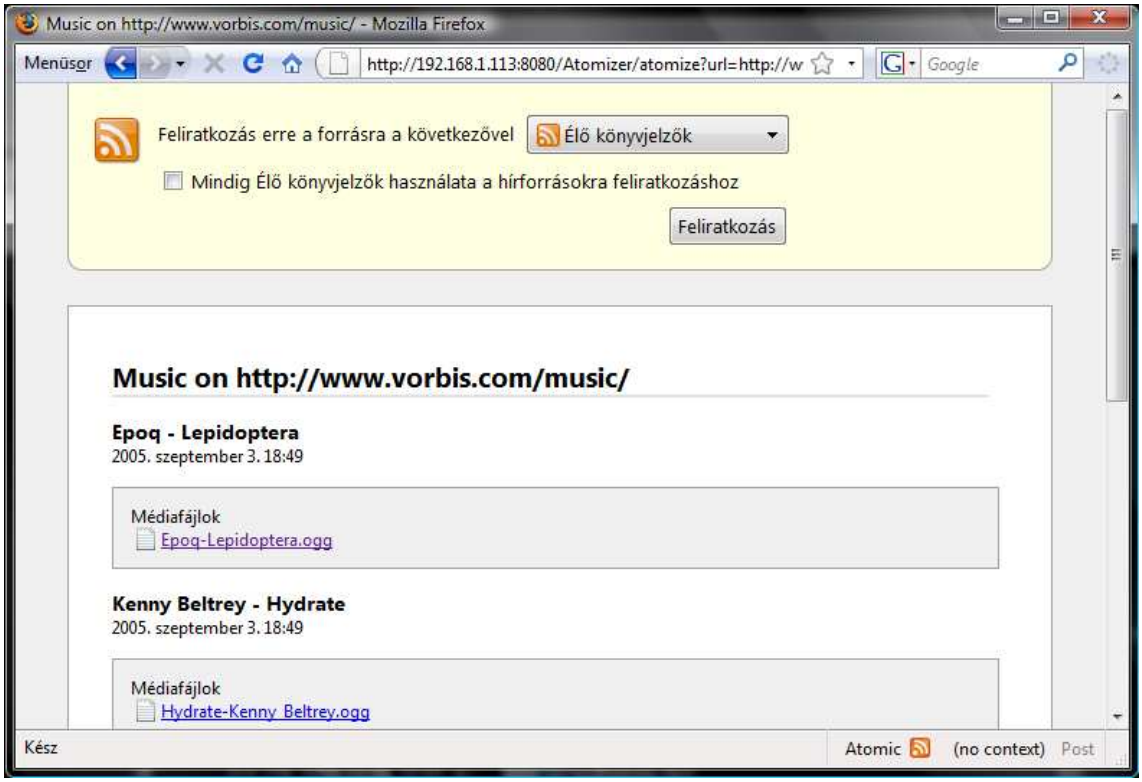
A munkámhoz egy kiváló nyílt forráskódú fejlesztői környezetet választottam, az Eclipse-et, több változata közül is a Java Enterprise Edition változatához megfelelőt. Ezt az IDE-t találtam a legkézreállóbb megoldásnak, rengeteg kényelmi funkcióval. Csak pozitív jelzőkkel tudom illetni az Eclipse-et, nagyon gyorsan lehetett vele dolgozni, és kiváló a támogatása a webes alkalmazások fejlesztéséhez, valamint a Tomcat-hez.

5.3. A konstrukció

Maga az alkalmazás egyetlen servletből áll, amely néhány, jól meghatározott feladattal és szerepkörrel rendelkező osztály működését koordinálja.

A `WebScraper` osztály szerepe a webes feladatok ellátása, ezek közé tartozik az egy weboldalon megjelenő linkek összegyűjtése, egy megadott URL-hez tartozó oldal adatainak lekérdezése, illetve ez végzi egy linklista elemein a megadott szűrő lefuttatását, azaz a lista szűrését.

A linkek összegyűjtése a következő módon történik. Egy JTidy-objektum létrehozásával megkezdődik az URL-ről lekért tartalom szabályos alakra hozása és abból DOM-fa építése. Ebben a DOM-fában XPath kifejezés használatával kikereshetők azok az `a` elemek, amelyek rendelkeznek `href` attribútummal. Ezekből



5.3. ábra. A podcast nézete Mozilla Firefoxban

linkeket készítünk.

A linkek a `Href` osztály egyes példányaiban tárolódnak, amelyben megjelenik a linkek URL-je, a hivatkozás szövege, illetve emellett tárolható későbbiekben a hivatkozott tartalom médiatípusa, mérete, kódolása, frissítésének ideje, kivonata, illetve bármi más, amely a fejlécekből kideríthető.

A `MusicHrefFilter` osztály megvalósítja az általam definiált `HrefFilter` interfészt. Az interfész tetszőleges linkszűrő létrehozására lehetőséget ad, abban a konkrét esetben, amikor a `WebScrapernek` a `MusicHrefFilter` egy példányát adjuk át szűrésre, az a linkeket aszerint szűri le, hogy a hivatkozott URL-en található tartalom zenei típusú vagy sem. Ezt úgy tudja meg, hogy a `WebScraper` segítségével egy HTTP HEAD-kérést küld az URL-re, a kérésre adott válaszban így csak fejlécadatok szerepelnek, s azok között a webes erőforrás médiatípusa is. Ezt, és még néhány adatot eltárol az átadott link példányában. Maga a filter nem kezel listát, csak egy linkről állapítja meg, hogy a listába tartozik-e vagy sem, illetve további adatokkal látja el, ha szükséges. A lista szűréséről a `WebScraper` gondoskodik, amely egyenként nézi végig a linkeket.

Amint elkészül a lista a zenefájlokat címző linkekről, a `FeedGenerator` kapja

meg a vezérlést, és előállítja a feed XML-szövegét. Ehhez felhasználja a feedelemeknél a link szövegét a feedelem címeként, a hivatkozott tartalom utolsó módosítási idejét a feedelem utolsó módosítási idejeként, a tartalom URL-jét és médiatípusát a feedelem enclosure típusú linkjeként és a tartalmaként is, hivatkozva. A feed módosítási ideje a „legfiatalalabb" tartalom ideje lesz.

Összefoglalva: a servlet összegyűjti a linkeket a `WebScraper` segítségével, megszűri és adatokkal látja el a linkeket a `WebScraper` és a `MusicHrefFilter` segítségével, és előállítja a feedet a `FeedGenerator` segítségével. A végeredmény a kezdeményező ágensnél megjelenik.

A servletnek az „atomizálni" kívánt oldal URL-jét GET-paraméterként kell átadni, a paraméter neve `url`. Ennek megkönnyítésére a webalkalmazás kontextusának gyökerében egy HTML-lapon egy űrlapon is megadható.

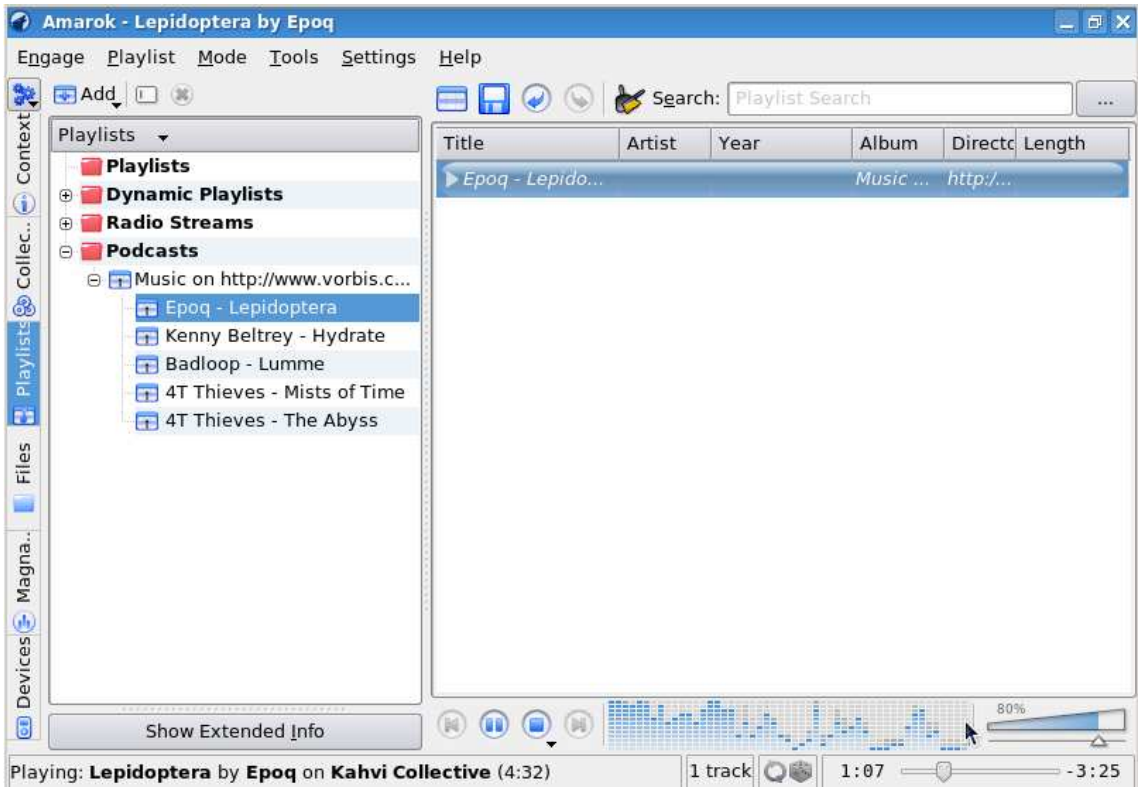
5.4. Az eredmények

Az Atomizer kimenetét egy azt megérteni képes aggregátorban lehet felhasználni, többek között ilyen aggregátorként használható az Amarok nevű zenelejátszó program. Ezt szemlélteti az 5.1. és az 5.4. ábra (a 47. és az 52. oldalon). Az eredeti oldalt mutatja az 5.2. ábra, az eredményezett feed képét a böngészőben pedig az 5.3. ábrán láthatja a kedves olvasó. Ez utóbbi képen a böngésző címsorában lévő URL-t kell bejegyezni az aggregátorba, hogy a kívánt oldalról szóló feedet kapja. Az aggregátor ezután rendszeresen frissíti azt, és így nyilvánvalóan az oldal tartalmának megfelelő lesz a feed is.

Teljesen nyilvánvaló, hogy ez a nagyon egyszerű alkalmazás csupán szemléltetni kívánja: egy egyszerű formátum milyen sokoldalúan használható fel. Az ötlet emögött az, hogy néhány, elsőre egymástól távolinak tűnő alkalmazást hozunk közelebb egy közös platformmal, a webbel, és – talán nyugodtan kijelenthető – annak nyelvével, az XML-lel.

Az Atomizer jelenlegi állapotában nem használ adatbázist *back-end*-ként, jóllehet profitálhatna belőle. Szélesebb körű, komoly felhasználása esetén problémát jelentene, hogy az atomizált oldalt minden alkalommal újra letölti. Ez elkerülhető, ha egy adatbázisban tárolja egy oldal korábbi lekérésének eredményét, és csak HEAD-kérést küldene az oldalnak, így, ha az nem változott, a korábbi feedet adhatja vissza (cache-elhet).

A fejlesztés melléktermékeként az eredeti témámtól eltérő tapasztalatot is sze-



5.4. ábra. Amarok: az első feedelem lejátszása

reztem, nevezetesen a weboldalak bejárásával és azokon megjelenő adatok tárolásával kapcsolatban.

6. fejezet

Összefoglalás

Kétségtelen, hogy a dolgozatban bemutatott elvek, formátumok és protokollok legszéleskörűbb alkalmazói a blogok. Ez nem is véletlen, a blogok éppen azokat az írásokat jelentik, amelyeket írójuk meg akar osztani a nagyvilággal. Megosztani azonban nem csak személyes irományokat, esetleg híreket, hanem mint láttuk, zeneszámokat, fényképeket, tulajdonképpen bármilyen digitálisan leírható dolgot is lehet.

Az RSS és az Atom arra adott megoldást, hogy a megosztott epizodikus tartalmakat egyszerűen tudjuk követni, anélkül, hogy az időnket fecséreljük felesleges ellenőrzésekkel. Ráadásként több forrásból egyszerre is követhetjük a történéseket, integráltan, egy helyen. Az XML, bár ember által is olvasható formátum, alapvetően nem emberi szemnek íródott – a számítógép teszi értelmezhetővé, vagy maga értelmezi: segítségével automatizálható nem csak a hírek egy helyre gyűjtése, hanem például két együttműködő weboldal tartalmi frissítése is.

Az Atom publikációs protokollja lehetővé tette ilyen tartalmak létrehozását egyszerű programok segítségével. A szabványos interfész és az egyszerűsége miatt gyorsan bevezethető szinte bárhol, ahol hírelemek létrehozása kívánatos. A felhasználható API-k száma napról napra nő.

Az idő múlásával és a tapasztalatok összegyűjtésével mostanra viszonylag stabilnak tekinthető az informatikának ezen kis szegmense. Úgy tűnik, a kezdeti nehézségeken (gondoljunk az RSS kalandos történetére) lassan sikerül túljutni; bár ne feledkezzünk meg a formátumok nyitottságáról, hiszen továbbfejleszhetőek maradnak kiterjesztési pontjaikon.

Irodalomjegyzék

[1] Tim O'Reilly: What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software
http://www.oreillynet.com/lpt/a/6228

[2] Kevin Kelly: We Are the Web
http://www.wired.com/wired/archive/13.08/tech_pr.html

[3] Mark Pilgrim: The myth of RSS compatibility
http://diveintomark.org/archives/2004/02/04/incompatible-rss

[4] http://www.rssboard.org/rss-specification

[5] http://www.ietf.org/rfc/rfc4287.txt

[6] http://www.ietf.org/rfc/rfc5023.txt

[7] Roy Thomas Fielding: Architectural Styles and the Design of Network-based Software Architectures (Doctoral dissertation), University of California, Irvine, 2000. 5. fejezet.
http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf

[8] Sam Ruby: Anatomy of a Well Formed Log Entry
http://www.intertwingly.net/blog/1472.html

[9] RSS 2.0 and Atom Compared
http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared

[10] Syndication Format Family Tree
http://en.wikipedia.org/wiki/Syndication_format_family_tree

[11] http://code.google.com/apis/gdata/

[12] https://rome.dev.java.net/

[13] http://www.atomenabled.org/

[14] http://www.fsp.org/