Webontológiák felhasználási lehetőségei

Doktori (PhD) értekezés

Jeszenszky Péter

Témavezető: Dr. Bognár Katalin

Debreceni Egyetem
Természettudományi Doktori Tanács
Informatikai Tudományok Doktori Iskola

Debrecen, 2010

Ezen értekezést a Debreceni Egyetem Természettudományi Doktori Tanács Informatikai Tudományok Doktori Iskola Alkalmazott információ technológia és elméleti háttere programja keretében készítettem a Debreceni Egyetem természettudományi doktori (PhD) fokozatának elnyerése céljából.

Debrecen, 2010. december 1.

Jeszenszky Péter
doktorjelölt

Tanúsítom, hogy Jeszenszky Péter doktorjelölt 2004–2007 között a fent megnevezett Doktori Iskola Alkalmazott információ technológia és elméleti háttere programjának keretében irányításommal végezte munkáját. Az értekezésben foglalt eredményekhez a jelölt önálló alkotó tevékenységével meghatározóan hozzájárult. Az értekezés elfogadását javasolom.

Debrecen, 2010. december 1.

Dr. Bognár Katalin
témavezető

Webontológiák felhasználási lehetőségei

Értekezés a doktori (Ph.D.) fokozat megszerzése érdekében
az informatika tudományágban

Írta: Jeszenszky Péter okleveles programtervező matematikus

Készült a Debreceni Egyetem
Informatikai Tudományok Doktori Iskolája
(TODO programja) keretében

Témavezető: Dr. Bognár Katalin

A doktori szigorlati bizottság:

elnök:	Dr. .....
tagok:	Dr. .....
	Dr. .....

A doktori szigorlat időpontja: 200... . ...

Az értekezés bírálói:

Dr. .....
Dr. .....
Dr. .....

A bírálóbizottság:

elnök:	Dr. .....
tagok:	Dr. .....
	Dr. .....
	Dr. .....
	Dr. .....

Az értekezés védésének időpontja: 200... . ...

Tartalomjegyzék

1. fejezet

Bevezetés

Tim Berners-Lee 1989-ben vázolta elképzelését egy olyan globális információs rendszerről, amely egy évvel később a World Wide Web nevet kapta a keresztségben [5]. Maga a megvalósítás is teljes egészében az ötletgazdától származik, így Tim Berners-Lee a tervezője a HTTP, HTML és URI néven közismert szabványoknak. Szerzője továbbá az első webszervernek, böngésző és weboldal szerkesztő programnak. (A World Wide Web történetéről lásd Tim Berners Lee [4] könyvét.)

A elmúlt húsz évben a World Wide Web természetes módon ment át egy olyan fejlődésen, amelynek kapcsán szokás a Web 1.0 és Web 2.0 kifejezésekkel jellemzett életszakaszokat megkülönböztetni. A Web 1.0 elnevezés a web kétezres évek elejéig tartó életszakaszára használt. Egy átmenet jellemzi a statikus webtől egy dinamikus web felé: míg a korai webet statikus dokumentumok alkották, a kétezres évek webjén már mindennapos a dinamikusan előállított weboldalak használata.

A Web 2.0 nem a World Wide Web egy új verzióját jelenti, hanem egy olyan webet, amelyen kiemelt szerepet kap az információk megosztása, a közösségek és az együttműködés. Olyan információs szolgáltatások jellemzik, mint például a blogok, wikik, közösségi hálózatok és webszolgáltatások. A felhasználók nem csupán passzív fogyasztói a Web 2.0 webhelyek kínálta tartalomnak, hanem részt vehetnek annak kialakításában. A szolgáltató feladata sok esetben csupán az infrastruktúra biztosítása és a teljes tartalmat egy közösség állítja elő. Ráadásul a tartalom létrehozásához nem szükséges speciális szaktudás sem, gondoljunk csak például a blogokra vagy wikikre! A megvalósítás oldaláról a Web 2.0-t olyan komplex technológiai háttér használata jellemzi, amely lehetővé teszi az asztali alkalmazásokra egyre jobban hasonlító webes alkalmazások létrehozását.

A web természetes evolúciójának egy lehetséges következő szintje a **szemantikus web**, amely az információk automatikus feldolgozhatóságát he-

lyezi középpontba. Az elgondolás Tim Berners-Lee-től származik. A fogalmat a nagyközönség számára egy 2001-ben a Scientific American folyóiratban megjelent cikk [6] tette közismertté.

Noha a Web 2.0 egyik alapeszközét jelentik a webszolgáltatások, amelyekkel gépi feldolgozásra szánt XML formátumban lehet strukturált adatokhoz jutni, a weben elérhető tartalom jelentős része ma is emberi fogyasztásra szánt szöveg. A weboldalak nyelveként szolgáló HTML és XHTML ugyan jelölőnyelvek, amelyek lehetővé teszik bizonyos szövegrészek azonosítását, de alapvetően a dokumentumok megjelenítéséhez használtak. Éppen ezért nagyon nehéz feladat a weboldalakból a megfelelő információk automatikus kinyerése.

A szemantika hiánya egy olyan probléma, amelyre Tim Berners-Lee már egy 1994-es előadásban felhívta a figyelmet [3]. A szemantikus web lesz majd az a környezet, amelyben az adatok jelentést nyerhetnek. Ma már kiforrottnak lehet tekinteni az alapjául szolgáló olyan alapvető szabványokat, mint az RDF [2], OWL [1] és SPARQL [7], amelyek gyakorlati alkalmazhatóságát számos felhasználás bizonyítja. Ráadásul a napjainkban reneszánszát élő Linked Data alkalmazások képében sokak szerint megvalósulni látszik a szemantikus web.

Az értekezés szerzőjét elsősorban az a kérdés motiválta, hogy miként lehet szemantikus web alkalmazások által is kiaknázhatóvá tenni a weben elérhető legkülönfélébb erőforrásokat. Eredményként a szerző több formátumhoz alkotott állományok RDF-ben történő leírásához alkalmas OWL ontológiákat. A dolgozat a szerző a szemantikus webhez kapcsolódó ezen saját eredményeit tárgyalja.

A Modellezés című I. rész egy a szerző által listák ábrázolásához kifejlesztett OWL webontológiát mutat be, amely az ontológiatervezés egy „melléktermékeként” született.

A II. második rész a szerző az RDF kinyeréssel kapcsolatos munkáit tartalmazza. 3. fejezete néhány RDF kinyerő konverziós programot, valamint egy ezek használatát egységesítő keretrendszert ismertet. 4. fejezetének témája pedig egy olyan a népszerű Firefox böngészőhöz fejlesztett egyedi funkció, amellyel erőforrásokba beágyazott metaadatokat lehet vizsgálni.

A Csomagkezelés című III. rész egy szoftvercsomag metaadatokat közzétevő Linked Data szolgáltatást mutat be, valamint ehhez kapcsolódva szoftvercsomagok modellezésére szolgáló OWL webontológiákat. Bevezetésként helyett kapott a részben egy áttekintés a csomagkezelésről (5. fejezet), és egy a szerző által a hazai közönség számára hiánypótlónak szánt Linked Data áttekintés (6. fejezet).

Az RPM csomagformátumnak a szerző egy külön függeléket szentelt, mivel az ebben leírtak a dolgozat több részében is hivatkozásra kerülnek. A

Linked Data kapcsán került egy olyan függelék az értekezés végére, amely az erőforrások azonosításával foglalkozik.

Irodalomjegyzék

[1] Web Ontology Language (OWL). URL http://www.w3.org/2004/OWL/.

[2] Resource Description Framework (RDF). URL http://www.w3.org/RDF/.

[3] Tim Berners-Lee. Plenary talk at WWWF94, 1994. URL http://www.w3.org/Talks/WWW94Tim/.

[4] Tim Berners-Lee. *Weaving the Web: The original design and ultimate destiny of the World Wide Web, by its inventor*. Harper, 1999.

[5] Tim Berners-Lee. Information management: A proposal, 1989. URL http://www.w3.org/History/1989/proposal.html.

[6] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[7] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. URL http://www.w3.org/TR/rdf-sparql-query/.

I. rész

Modellezés

2. fejezet

Listák modellezése az OWL-ben

2.1. Bevezetés

Bármilyen meglepően hangzik, annak ellenére, hogy listaszerű szerkezetek kezelése szinte minden programozási nyelvben megvalósítható, elég mostohán alakult a listák sorsa az RDF és OWL kapcsán. Állításunkat a 2.2. és 2.3. szakaszokban támasztjuk alá, megvizsgálva az RDF és OWL listák modellezéséhez alkalmas lehetőségeit valamint az ezekkel kapcsolatban felmerülő problémákat.

A 2.4. szakaszban egy a szerző által kidolgozott általános tervezési minta kerül bemutatásra, amely lehetővé teszi listaszerű szerkezetek használatát OWL ontológiákban. A konstrukció egyaránt alkalmas kizárólag literálokat, kizárólag egyedeket, valamint literálokat és egyedeket vegyesen tartalmazó listákhoz, támogatja továbbá tipizált listák létrehozását is. A gyakorlatban alkalmazhatóságot szem előtt tartva a listák modellezése szigorúan az OWL 1 DL (egyben az OWL 2 DL) keretei között történik.

2.2. Listák modellezése az RDF-ben

Az RDF gyakorlati alkalmazásaiban gyakran kell listaszerű szerkezeteket ábrázolni. Ebben a szakaszban a szóba jöhető megoldásokat vesszük sorra, rámutatva ezek problémáira.

Listák ábrázolásához használhatunk RDF konténereket [21, 13, 18], amelyek tagok csoportjait reprezentáló speciális erőforrások. Leírásukhoz beépített osztályok és tulajdonságok állnak rendelkezésre. Az RDF három előre definiált konténertípust biztosít, az alternatíva-csoportot (Alt), a multihalmazt (Bag) és a sorozatot (Seq), amelyek közül az utóbbi alkalmas listák ábrázolásához. Konténerek használata sajnos több szempontból problémás:

- Míg a programozási nyelvekben létrehozni lehet a konténereket, addig az RDF-ben csupán leírni, azaz nincs lehetőség annak kijelentésére, hogy a konténereknek a felsorolt elemeken kívül nem lehetnek további tagjai is.

- Formális szemantika szempontjából a konténertípusok semmiben sem különböznek egymástól [18].

A fentiek közül az első hiányosság orvoslására kínálnak megoldást az RDF kollekciók, amelyek a Lisp és vele rokon programozási nyelvek listáihoz hasonló szerkezetek. Leírásukhoz egy beépített szókészlet áll rendelkezésre. A kollekciók a konténerektől eltérően zártak abban az értelemben, hogy nem tartalmazhatnak további tagokat a leírásukban felsoroltakon túl.

A konténerek és kollekciók használatát nem javasolják Linked Data alkalmazásokban (lásd például a témában alapvetőnek számító [12] dokumentum állásfoglalását). Ennek egyik oka az, hogy ábrázolásuk tipikusan üres csomópontokkal történik, amelyekre a tartalmazó gráfokon kívülről lehetetlen RDF linkekben hivatkozni, ráadásul a különböző forrásokból származó adatok összefésülését is megnehezítik. Másrészt jelenleg meglehetősen kényelmetlen a használatuk SPARQL [24] lekérdezésekben.[1]

A problémák elkerüléséhez megoldásként gyakran a konténert vagy kollekciót tartalmazó RDF kijelentés helyettesítését ajánlják az elemek számával egyező számú RDF kijelentéssel, amelyeket az eredeti kijelentésből úgy kapunk, hogy a konténer vagy kollekció helyére az egyes elemeket helyettesítjük be. Ezek a kijelentések együtt azonban mást jelenthetnek, mint az eredeti kijelentés, nem beszélve arról, hogy elvész a sorrend, amely sok esetben elfogadhatatlan.

2010 nyarán került megrendezésre a W3C szervezésében az *RDF Next Steps* című workshop az RDF jövőjéről, amely után a W3C egy *The Future of RDF Standards* című nyilvános kérdőíven kérte ki a szakma véleményét az RDF esetleges későbbi továbbfejlesztéséről [19]. A kérdőívet összesen 126 a témával foglalkozó szakember töltötte ki, amelynek eredményei megtekinthetők a `http://www.w3.org/2002/09/wbs/1/rdf-2010/results` címen. Az RDF konténerek és kollekciók problémái számos véleményben visszaköszönnek, a visszajelzések ösztönözhetik a konténerek és kollekciók használatának esetleges jövőbeli újragondolását.

[1] A SPARQL lekérdező nyelv jelenleg fejlesztés alatt álló következő, 1.1 számú verziója [17] ezen a téren előrelépést jelent majd, lásd a szabvány részeként a [25] dokumentumban leírt *property path* mechanizmust.

2.3. Listák modellezése az OWL-ben

Ebben a szakaszban azt vizsgáljuk meg, hogy milyen lehetőségek adódnak OWL ontológiákban listaszerű szerkezetek megvalósítására és használatára. A gyakorlati alkalmazásokban sokszor csak adott típusú elemeket tartalmazó listákat kell kezelni, ezért tipizált listák implementálását tűzzük ki végcélul.

2.3.1. RDF konténerek és kollekciók használata

Noha az RDF konténerek és kollekciók több szempontból is problémát jelenthetnek, kézenfekvő választásnak tűnik ezek OWL ontológiákban használata. Sajnos azonban az OWL 1 [22] és OWL 2 [16] kedvező kiszámíthatósági tulajdonságokkal rendelkező OWL 1 DL és OWL 2 DL alnyelve sem teszi lehetővé a konténereket és kollekciókat reprezentáló osztályok ilyen szerepeltetését. Ha például egy tulajdonság értéktartományaként egy kollekció jelenik meg egy ontológiában, akkor az ontológiánk egy OWL Full ontológia. (OWL DL-ben a kollekciók csupán magának az ontológiának az ábrázolásához állnak rendelkezésre.)

Ha elfogadható számunkra az OWL Full, akkor a konténer és kollekció szókészletet felhasználva tipizált listákat is megvalósíthatunk. Erre például a Protégé-OWL FAQ [7] ad egy lehetséges megoldást.[2]

2.3.2. XML séma alapú megoldás

Azonos típusú atomi[3] értékekből álló listák kezeléséhez megoldásként szóba jöhet elméletileg egy alkalmas lista adattípus definiálása egy XML séma dokumentumban.

Az XML Schema: Component Designators [20] dokumentum mechanizmust biztosít sémakomponensek azonosításához, lehetővé téve így a sémákban definiált adattípusokra történő külső hivatkozást URI-k formájában. A lista adattípusokat azonosító URI-k használhatók az RDF tipizált literáljaiban is. Az adattípus lexikális terét az elemtípus literáljaiból álló karaktersorozatok alkotják, amelyekben az elemeket szóköz karakterek választják el egymástól.

[2] A megoldást bemutató példa a következő címen érhető el: `http://protege.stanford.edu/plugins/owl/testdata/list-example.owl`

[3] Az „atomi” jelző egészen pontosan az XML Schema [26, 11] úgynevezett atomi adattípusait jelenti. Az atomi adattípusok értékterének elemeit a szabvány oszthatatlanoknak tekinti. A beépített adattípusok közül ilyenek például az `xsd:date`, `xsd:double`, `xsd:integer` és `xsd:string`. Nem atomi típusok a lista és unió adattípusok.

Sajnos sem az OWL 1, sem az OWL 2 nem támogatja az ilyen módon definiált adattípusokat (lásd a szabványok a nem támogatott adattípusok kezelésére vonatkozó részeit). Problémát jelenthet az is, hogy bizonyos atomi típusok – például az `xsd:string` típus – literáljaiban megengedettek szóköz karakterek, amely tény lehetetlenné teheti a lexikális forma megfelelő elemekre bontását. Ráadásul [20] jelenleg még nem került elfogadásra W3C ajánlásként, így nem tekinthető stabil szabványnak.

2.3.3. Listák megvalósítása saját osztályokkal és tulajdonságokkal

Ha nem kívánjuk átlépni az OWL DL megszabta kereteket, akkor mindenképpen megfelelő saját osztályokat és tulajdonságokat kell definiálnunk a listák modellezéséhez. [15] egy az OWL DL határain belül maradó általános megoldást ismertet. A széles körben elterjedt szabad és nyílt forrású Protégé ontológia-szerkesztő [6] 3.x számú verzióihoz áll rendelkezésre egy olyan OWL Wizards nevű bővítmény [4], amely felhasználóbarát módon teszi lehetővé a cikkben vázolt tervezési mintának megfelelő listák létrehozását.

A tervezési mintához a 2.1. ábrán látható osztályok és tulajdonságok használtak.[4] A konstrukció alapján tipizált listák leírására szolgáló osztályok létrehozása olyan `owl:allValuesFrom` tulajdonságkorlátozásokkal lehetséges, amelyekben a `hasContents` és az `isFollowedBy` tulajdonságok értéktartományát megfelelően korlátozzuk.

A megoldás komoly fogyatékossága, hogy OWL DL-ben kizárólag egyedekből álló listák létrehozásához használható. Ennek oka az, hogy az első elem megadására szolgáló `hasContents` tulajdonság egyedtulajdonság. Sem az OWL 1 DL, sem pedig az OWL 2 DL nem tekinti egyedeknek a literálokat, így a fenti konstrukció értelemszerűen alkalmatlan literálokat is tartalmazó listákhoz. (OWL DL-ben literálok nem megengedettek egyedtulajdonságokat tartalmazó kijelentések tárgyaként.)

A Collections Ontology [14] egy a [15] munkán alapuló OWL 1 DL webontológia halmazok, multihalmazok és listák kezeléséhez. Definiál ugyan egy az elemek számát szolgáltató adattípus-tulajdonságot, azonban ehhez csupán értelmezési tartományt és értékkészletet határoz meg, nem követeli meg, hogy értéke valóban az elemek számával legyen azonos. Nem teszi lehetővé az OWL DL-en belül maradva elemekként literálok használatát sem.

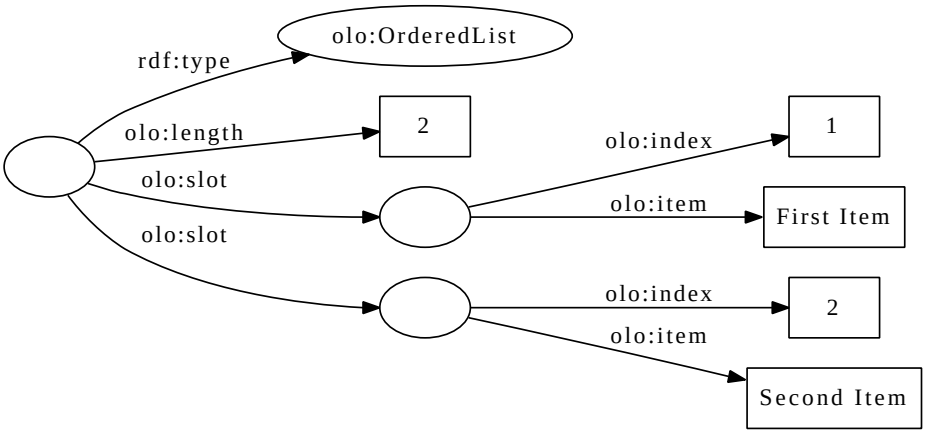
Az előbbiektől lényegesen különböző megoldást biztosít listák kezeléséhez az Ordered List Ontology [9], amelynek gyakorlati alkalmazását a 2.2. ábra

[4] A `http://www.co-ode.org/ontologies/lists/` címen érhető el a tartalmazó ontológia, amelyre továbbiakban CO-ODE ontológiaként hivatkozunk.

```
Declaration(ObjectProperty(list:hasListProperty))
ObjectPropertyDomain(list:hasListProperty list:OWLList)

Declaration(ObjectProperty(list:hasContents))
SubObjectPropertyOf(list:hasContents list:hasListProperty)
FunctionalObjectProperty(list:hasContents)

Declaration(ObjectProperty(list:isFollowedBy))
SubObjectPropertyOf(list:isFollowedBy list:hasListProperty)
TransitiveObjectProperty(list:isFollowedBy)
ObjectPropertyRange(list:isFollowedBy list:OWLList)

Declaration(ObjectProperty(list:hasNext))
SubObjectPropertyOf(list:hasNext list:isFollowedBy)
FunctionalObjectProperty(list:hasNext)

Declaration(Class(list:EmptyList))
EquivalentClasses(
  list:EmptyList
  ObjectIntersectionOf(
    ObjectComplementOf(
      ObjectSomeValuesFrom(list:hasContents owl:Thing)
    )
    list:OWLList
  )
)
EquivalentClasses(
  list:EmptyList
  ObjectIntersectionOf(
    ObjectComplementOf(
      ObjectSomeValuesFrom(list:isFollowedBy owl:Thing)
    )
    list:OWLList
  )
)

Declaration(Class(list:OWLList))
SubClassOf(list:OWLList
  ObjectAllValuesFrom(list:isFollowedBy list:OWLList))
```

2.1. ábra. A CO-ODE ontológia osztályai és tulajdonságai listák megvalósításához

szemlélteti. Látható, hogy a listákat olyan „rekeszek” alkotják, amelyek mindegyikéhez egy elem és egy sorszám tartozik. A számozás kapcsán azonban mindössze annyit követel az ontológia, hogy minden rekesznek pontosan egy nemnegatív sorszáma legyen, nem definiál tehát megfelelő szemantikát. Az ontológia ráadásul OWL Full-ban van.



(a)

```
@prefix : <http://purl.org/ontology/olo/core#>  .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[]    a :OrderedList ;
      :length 2 ;
      :slot [
             :index 1 ;
             :item "First Item"^^xsd:string ;
      ] ;
      :slot [
             :index 2 ;
             :item "Second Item"^^xsd:string ;
      ] .
```

(b)

2.2. ábra. Ordered List Ontology: kételemű lista ábrázolása gráfként és Turtle [10] szintaxissal

Jelenleg sajnos nem áll rendelkezésre olyan megfelelően kidolgozott és azonnali használatra kész tervezési minta, amely megnyugtató választ adna a fenti megoldások hiányosságaira az OWL DL-en belül.

A listákkal kapcsolatos problémák szóba kerültek az OWL 2 kidolgozása során is, erről lásd a [8] wiki oldalt.

2.4. A javasolt tervezési minta

2.4.1. Megvalósítás

A szerző által adott megoldás Boris Motik az OWL 2 számára javasolt ötletének kidolgozása. Boris Motik felvetését lásd a [23] címen elérhető levelezési lista üzenetben és az ennek kapcsán kibontakozott párbeszédben. Motik az OWL 2 szókészletének bővítését javasolta listák ábrázolásához alkalmas osztályokkal és tulajdonságokkal, amely javaslat végül nem került adaptálásra a szabványban.

A tervezési minta megvalósítását tartalmazó OWL ontológia a `http://purl.org/net/vocabulary/list.owl` címen érhető el, amely technikai okokból a 2.3. és 2.4. ábrán látható az OWL 2 funkcionális szintaxisával ábrázolva.

A listákat a `List` osztály egyedei reprezentálják, amelyek első elemét a `hasIndividual` vagy a `hasLiteral` tulajdonság adja meg, amelyekre előírjuk azt, hogy a `List` osztály egy adott egyedénél a kettő közül csak az egyik használható. Megszokott módon a `hasNext` tulajdonság adja meg a lista további elemeit tartalmazó listát. Az osztály segítségével leírható listák értelemszerűen egyedeket és literálokat is tartalmazhatnak.

Mivel `hasIndividual` egyedtulajdonság, `hasLiteral` pedig adattípustulajdonság, sajnos az OWL DL keretein belül nincs lehetőségünk egy olyan főtulajdonság definiálására, amelynek mindkét tulajdonság altulajdonsága.

Az `EmptyList` osztály reprezentálja az üres listát. Ebbe az osztályba csak egyetlen egyed tartozik, amelyet a `http://purl.org/net/vocabulary/list.owl#nil` URI azonosít.

Csak egyedeket vagy literálokat tartalmazó listák ábrázolására szolgálnak az ontológia `LiteralList` és `IndividualList` osztályai, amelyek egyedeinél az első elem megadásához a `hasIndividual` és a `hasLiteral` tulajdonságok közül csak a megfelelő használható. Egy további megszorítás, hogy az összes többi elemet tartalmazó és a `hasNext` tulajdonsággal megadott lista megfelelő típusú kell hogy legyen.

A 2.5. ábra mutatja a listaosztályok hierarchiáját, amelyen megfigyelhető, hogy az üres listát reprezentáló `EmptyList` osztály implicit módon alosztálya a `LiteralList` és `IndividualList` osztályoknak is.

2.4.2. Tipizált listák létrehozása

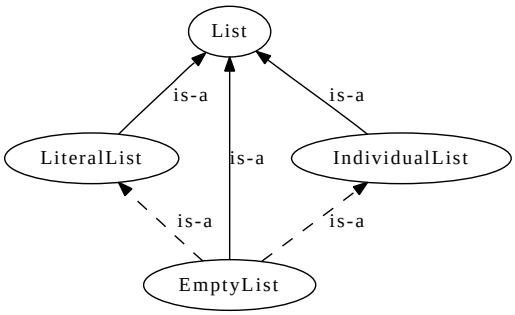
A fenti modellben egyszerű olyan tipizált listákhoz megfelelő osztályokat definiálni, amelyek csak egy adott típusú elemeket tartalmazhatnak. Ehhez létre kell hozzuk az `IndividualList` vagy `LiteralList` osztály egy olyan

```
Prefix(xsd:=<http://www.w3.org/2001/XMLSchema#>)
Prefix(owl:=<http://www.w3.org/2002/07/owl#>)
Prefix(list:=<http://purl.org/net/vocabulary/list.owl#>)

Ontology(
  <http://purl.org/net/vocabulary/list.owl>

  Declaration(ObjectProperty(list:hasIndividual))
  FunctionalObjectProperty(list:hasIndividual)
  ObjectPropertyDomain(list:hasIndividual list:List)

  Declaration(DataProperty(list:hasLiteral))
  FunctionalDataProperty(list:hasLiteral)
  DataPropertyDomain(list:hasLiteral list:List)

  Declaration(ObjectProperty(list:isFollowedBy))
  TransitiveObjectProperty(list:isFollowedBy)

  Declaration(ObjectProperty(list:hasNext))
  SubObjectPropertyOf(list:hasNext list:isFollowedBy)
  FunctionalObjectProperty(list:hasNext)
  ObjectPropertyDomain(list:hasNext list:List)
  ObjectPropertyRange(list:hasNext list:List)

  Declaration(Class(list:List))
  SubClassOf(list:List owl:Thing)
  SubClassOf(
    list:List
    ObjectComplementOf(
      ObjectIntersectionOf(
        DataExactCardinality(1 list:hasLiteral)
        ObjectExactCardinality(1 list:hasIndividual)
      )
    )
  )
```

2.3. ábra. A tervezési mintához készült OWL DL ontológia

```
  Declaration(Class(list:EmptyList))
  EquivalentClasses(
    list:EmptyList
    ObjectIntersectionOf(
      list:List
      ObjectExactCardinality(0 list:hasIndividual)
      ObjectExactCardinality(0 list:hasNext)
      DataExactCardinality(0 list:hasLiteral)
    )
  )
  ClassAssertion(list:EmptyList list:nil)
  EquivalentClasses(list:EmptyList ObjectOneOf(list:nil))

  Declaration(Class(list:IndividualList))
  EquivalentClasses(
    list:IndividualList
    ObjectIntersectionOf(
      list:List
      ObjectAllValuesFrom(list:hasNext list:IndividualList)
      DataExactCardinality(0 list:hasLiteral)
    )
  )

  Declaration(Class(list:LiteralList))
  EquivalentClasses(
    list:LiteralList
    ObjectIntersectionOf(
      list:List
      ObjectAllValuesFrom(list:hasNext list:LiteralList)
      ObjectExactCardinality(0 list:hasIndividual)
    )
  )

)
```

2.4. ábra. A tervezési mintához készült OWL DL ontológia (folytatás)



2.5. ábra. A listaosztályok hierarchiája (szaggatott vonalak jelzik az implicit, kikövetkeztethető kapcsolatokat)

alosztályát, amelynél `owl:allValuesFrom` tulajdonságkorlátozásokkal alkalmas módon szorítjuk meg a `hasIndividual`/`hasLiteral` és `hasNext` tulajdonságok használatát.

A gyakorlati megvalósítást egy példában mutatjuk be a 2.6. ábrán. Itt egy olyan `IntegerList` osztályt definiálunk, amely `xsd:integer` típusú literálokból álló listák kezelését teszi lehetővé. A 2.6(a). ábra tartalmazza az ontológiához hozzáadandó konstrukciót az OWL 2 funkcionális szintaxisával megadva. A 2.6(b). ábrán látható, hogy az `IntegerList` osztály hogyan illeszkedik az osztályhierarchiába.

2.4.3. Példák a konstrukció használatára

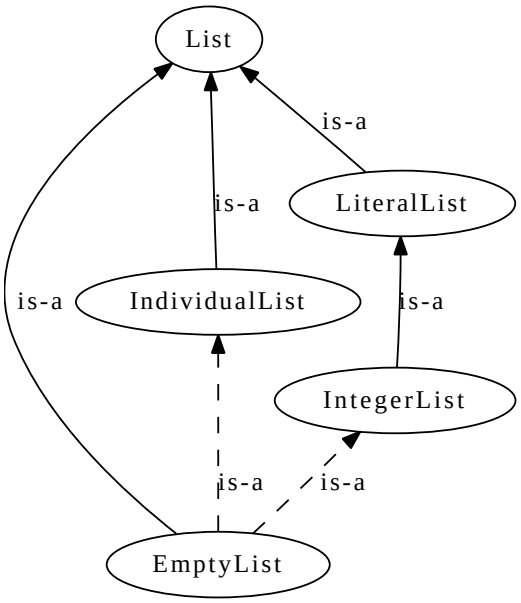
Ebben a szakaszban néhány példát mutatunk be az ontológiában definiált osztályok és tulajdonságok használatára.

A 2.7. ábra egy háromelemű, `xsd:string` típusú literálokból álló listának a szókészlettel leírását szemlélteti. Szembetűnő, hogy az egyes részlistákat reprezentáló üres csomópontok típusa nincs explicit módon megadva. Az ontológián alapuló következtetés révén kinyerhető azonban a modellből, hogy mindhárom üres csomópont a `LiteralList` osztály példánya. Amennyiben az `IntegerList` osztály mintájára az `xsd:string` típushoz létrehozzuk a megfelelő `StringList` osztályt, akkor az üres csomópontok annak is implicit módon példányai lesznek.

Egy lista akár egyedeket és literálokat is tartalmazhat egyidejűleg, erre látható példa a 2.9. ábrán. A felső üres csomópont implicit módon a `List` osztály példánya, a másik kettő pedig a `LiteralList` osztályé. Amennyiben az `IntegerList` osztály mintájára az `xsd:date` típushoz létrehozzuk a megfelelő `DateList` osztályt, akkor az alsó üres csomópont annak is implicit módon példánya lesz.

```
Declaration(Class(list:IntegerList))
EquivalentClasses(
  list:IntegerList
  ObjectIntersectionOf(
    list:LiteralList
    ObjectAllValuesFrom(list:hasNext list:IntegerList)
    DataAllValuesFrom(list:hasLiteral xsd:integer)
  )
)
```

(a)



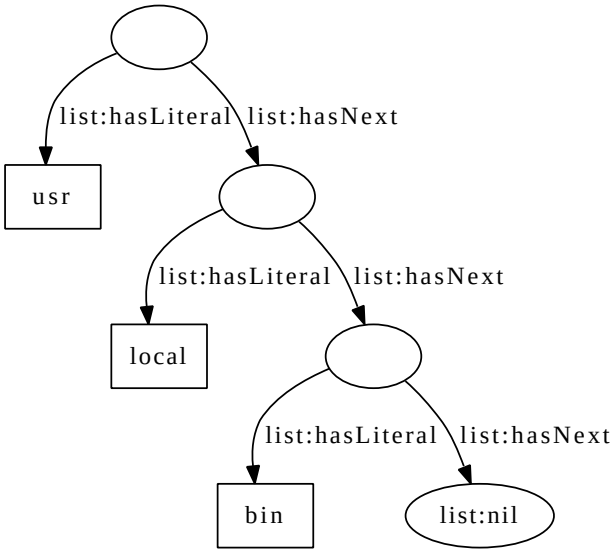
(b)

2.6. ábra. Új listaosztály definiálása. (a) A definíció az OWL 2 funkcionális szintaxisával ábrázolva. (b) Az osztály illeszkedése az osztályhierarchiába (szaggatott vonalak jelzik az implicit, kikövetkeztethető kapcsolatokat).

2.4.4. Az elemek számának korlátozása

Nem csak az elemek típusára, hanem azok számára is megfogalmazható megszorítás. Nyilvánvaló, hogy az

$$\mathrm{L}[n] = \begin{cases} \texttt{EmptyList}, & \text{ha } n = 0, \\ \texttt{ObjectSomeValuesFrom}(\texttt{hasNext } \mathrm{L}[n-1]), & \text{ha } n \geq 1 \end{cases}$$



(a)

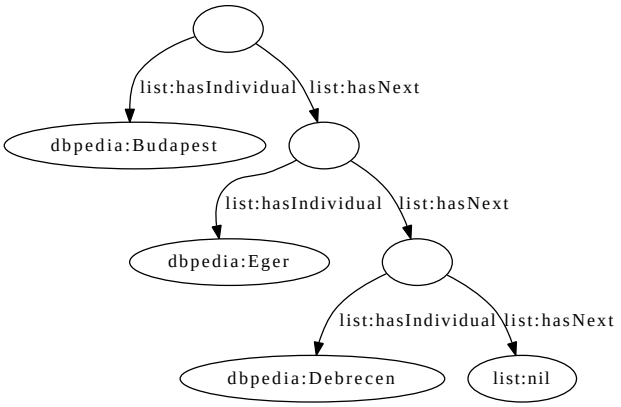
```
@prefix : <http://purl.org/net/vocabulary/list.owl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

[]    :hasLiteral "usr"^^xsd:string ;
      :hasNext
              [ :hasLiteral "local"^^xsd:string ;
                :hasNext
                        [ :hasLiteral "bin"^^xsd:string ;
                          :hasNext :nil
                        ]
              ] .
```

(b)

2.7. ábra. Literálokból álló lista ábrázolása gráfként és Turtle szintaxissal

kifejezés a pontosan n elemet tartalmazó listákat reprezentáló osztályt adja meg, szigorúan az OWL DL keretein belül maradva. A kifejezés értékeként adódó osztályt az `owl:equivalentClass` tulajdonsággal egy axiómában megfelelően el is nevezhetjük. Adott elemszámú tipizált listák kezeléséhez metszetképzést kell használni, amelynek gyakorlati megvalósítását a 2.10. ábra szemlélteti. Ezen egy olyan osztály definiálását láthatjuk, amelynek kiterjedése a pontosan öt egész számot tartalmazó listákból áll.

Az előbbiek mintájára írhatjuk fel a legalább adott számú elemet tartal-



(a)

```
@prefix : <http://purl.org/net/vocabulary/list.owl#> .
@prefix dbpedia: <http://dbpedia.org/resource/> .

[]    :hasIndividual dbpedia:Budapest ;
      :hasNext
              [ :hasIndividual dbpedia:Eger ;
                :hasNext
                        [ :hasIndividual dbpedia:Debrecen ;
                          :hasNext :nil
                        ]
              ] .
```

(b)

2.8. ábra. Egyedekből álló lista ábrázolása gráfként és Turtle szintaxissal

mazó listákhoz szükséges OWL DL konstrukciót. Az

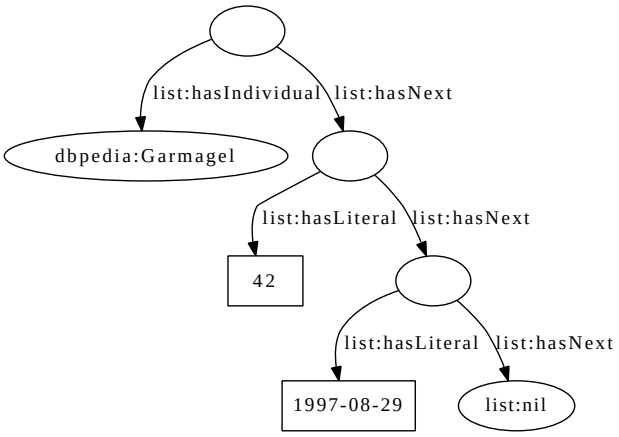
$$\mathrm{L}'[n] = \begin{cases} \texttt{List}, & \text{ha } n = 0, \\ \texttt{ObjectSomeValuesFrom}(\texttt{isFollowedBy}\ \mathrm{L}[n-1]), & \text{ha } n \geq 1 \end{cases}$$

kifejezés egy olyan osztályt ír le, amelynek kiterjedésébe a legalább n elemű listák tartoznak.

Legfeljebb $n > 1$ elemű listákhoz az előbbi formula mintájára kézenfekvő módon kínálja magát az

$$\texttt{ObjectIntersectionOf}(\texttt{List}\ \texttt{ObjectComplementOf}(\texttt{ObjectSomeValuesFrom}(\texttt{isFollowedBy}\ \mathrm{L}[n]))),$$

kifejezés, amely sajnos azonban a célhoz nem megfelelő az OWL nyílt világ feltételezése miatt. (Noha az `isFollowedBy` tulajdonság tranzitív, nem tranzitív lezárása a `hasNext` altulajdonságnak.) Kevésbé elegáns, de a helyes



(a)

```
@prefix : <http://purl.org/net/vocabulary/list.owl#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

[]     :hasIndividual <http://dbpedia.org/resource/Gargamel> ;
       :hasNext
               [ :hasLiteral "42"^^xsd:integer ;
                 :hasNext
                         [ :hasLiteral "1997-08-29"^^xsd:date ;
                           :hasNext :nil
                         ]
               ] .
```

(b)

2.9. ábra. Egyedekből és literálokból álló lista ábrázolása gráfként és Turtle szintaxissal

eredményt adja az alábbi kifejezés:

$$\mathrm{L}''[n] = \texttt{ObjectUnionOf}(\mathrm{L}[0] \ldots \mathrm{L}[n]), \qquad n \geq 0.$$

Vegyük észre, hogy noha a konstrukciókat az utóbbit kivéve egy rekurzív kifejezéssel írtuk le, ontológiákban a kifejezés adott n-re kifejtett értékét kell használni. Ez már kevés számú elem esetén is kényelmetlen és sok hibára lehetőséget adó feladat. Például ontológia-szerkesztőkben valósítható meg olyan funkció, amely felhasználóbarát módon teszi lehetővé a fenti minta alapján osztályok definiálását.

2.4.5. A konstrukció jellemzői

Az alábbiakban foglaljuk össze a bemutatott konstrukció legfontosabb jellemzőit:

```
Declaration(Class(list:ListOfFiveIntegers))
EquivalentClasses(
  list:ListOfFiveIntegers
  ObjectIntersectionOf(list:IntegerList L[5])
)
```

(a)

```
Declaration(Class(list:ListOfFiveIntegers))
EquivalentClasses(
  list:ListOfFiveIntegers
  ObjectSomeValuesFrom(list:isFollowedBy
    ObjectSomeValuesFrom(list:hasNext
      ObjectSomeValuesFrom(list:hasNext
        ObjectSomeValuesFrom(list:hasNext
          ObjectSomeValuesFrom(list:hasNext list:EmptyList)
        )
      )
    )
  )
)
```

(b)

2.10. ábra. Osztály definiálása adott elemszámú tipizált listához. Az (a) ábrán az L[5] kifejezés helyére a kifejezés értékét kell behelyettesíteni. A (b) ábrán látható L[5] kifejtésének eredménye.

- Szigorúan az OWL DL keretein belül marad. (Bonyolultsága egészen pontosan az $\mathcal{SHON}(D)$ kifejezéssel jellemezhető.)
- Lehetővé teszi kizárólag egyedeket, kizárólag literálokat, valamint egyedeket és literálokat vegyesen tartalmazó listák létrehozását.
- A `List` osztály valamennyi további listaosztály közös szuperosztálya.
- Az ismertetett módon teszi lehetővé további tipizált listaosztályok létrehozását.
- Alkalmas legalább, pontosan vagy legfeljebb adott számú elemet tartalmazó listák kezeléséhez.
- Az üres listát egyetlen egyed reprezentálja, amely implicit módon példánya a három előre definiált és minden további, a bemutatott módon megfelelően létrehozott listaosztálynak.

2.4.6. Szoftveres támogatás

A bemutatott konstrukció használatához támogatást biztosít a szerző által kifejlesztett szabad és nyílt forrású, a Jena Semantic Web Framework [2] keretrendszeren alapuló `OWLListUtils` [5] Java programkönyvtár.

A könyvtár fordításához és használatához Java fejlesztői környezet (Java SE 6 az ajánlott) valamint az Apache Maven 3 [3] telepítése szükséges. A Maven automatikusan elvégzi valamennyi szükséges függőség, így például a Jena Semantic Web Framework és a parancssori argumentumok feldolgozásához használt `Commons CLI` [1] osztálykönyvtár letöltését és telepítését.

Az osztálykönyvtár kényelmi eszközöket ad programozóknak a listaszerkezetek a Jena alkalmazói programozói interfészén keresztül történő manipulásához. Például a 2.7. ábrán látható lista létrehozása az alábbi kódrészlettel lehetséges:

```
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.RDFNode;

import hu.unideb.inf.owllistutils.jena.Util;

  Model        model;
  ...

  RDFNode list = Util.createList(
    new RDFNode[] {
```

```
      model.createTypedLiteral("usr"),
      model.createTypedLiteral("local"),
      model.createTypedLiteral("bin")
    }
  );

  // vagy

  RDFNode list = Util.createList(
    new String[] {"usr", "local", "bin"}
  );
```

Másrészt a csomag része egy olyan program, amely a bemutatott listaszerkezetekhez automatikusan előállítja a megfelelő OWL konstrukciókat. (Ezzel készültek az ábrákon látható listaosztályok definíciói.) A webontológia generátor programhoz jelenleg egy parancssoros felhasználói felület áll rendelkezésre, amelynek lehetőségeit alább láthatjuk:

```
usage: java hu.unideb.inf.owllistutils.tui.Main [options] <uri>
    --exactLength <n>      set exact number of elements
 -h,--help                 display this help and exit
    --import               import list ontology
 -iv,--individualList      subclass class IndividualList
 -l,--language <format>    write output in the language specified
                           (N-TRIPLES, TURTLE, RDF/XML,
                           RDF/XML-ABBREV, default: RDF/XML)
 -li,--literalList         subclass class LiteralList
    --maxLength <n>        set maximal number of elements
    --minLength <n>        set minimal number of elements
 -o,--output <file>        write output to the file specified instead
                           of standard output
 -t,--elementType <type>   element type (eg. xsd:integer,
                           http://xmlns.com/foaf/0.1/Person)
```

A program kötelező parancssori argumentumként várja a létrehozandó listaosztályt azonosító URI-t, opciókkal adható meg az elemek számára vonatkozó korlátozás és az elemek típusa.

Irodalomjegyzék

[1] Commons CLI. URL http://commons.apache.org/cli/.

[2] Jena Semantic Web Framework. URL http://jena.sourceforge.net/.

[3] Apache Maven. URL http://maven.apache.org/.

[4] OWL Wizards. URL http://www.co-ode.org/downloads/wizard/.

[5] OWLListUtils. URL http://www.inf.unideb.hu/~jeszy/OWLListUtils/.

[6] The Protégé Ontology Editor and Knowledge Acquisition System. URL http://protege.stanford.edu/.

[7] Protégé-OWL FAQ. URL http://protege.stanford.edu/doc/owl-faq.html.

[8] RDF list vocabulary. URL http://www.w3.org/2007/OWL/wiki/RDF_list_vocabulary.

[9] Samer A. Abdallah and Bob Ferris. Ordered List Ontology Specification, 2010. URL http://purl.org/ontology/olo/core#. version 0.72.

[10] David Beckett and Tim Berners-Lee. Turtle – Terse RDF Triple Language, 2008. URL http://www.w3.org/TeamSubmission/turtle/.

[11] Paul V. Biron and Ashok Malhotra. XML Schema Part 2: Datatypes Second Edition. W3C Recommendation, 2004. URL http://www.w3.org/TR/xmlschema-2/.

[12] Chris Bizer, Richard Cyganiak, and Tom Heath. How to Publish Linked Data on the Web. URL http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/.

[13] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004. URL http://www.w3.org/TR/rdf-schema/.

[14] Paolo Ciccarese. Collections Ontology Specification, 2009. URL http://swan.mindinformatics.org/spec/1.2/collections.html. revision 1.2.

[15] Nick Drummond, Alan Rector, Robert Stevens, Georgina Moulton, Matthew Horridge, Hai H. Wang, and Julian Seidenberg. Putting OWL in Order: Patterns for Sequences in OWL. In *OWL: Experiences and Directions*, 2006. URL http://www.webont.org/owled/2006/acceptedLong/submission_12.pdf.

[16] W3C OWL Working Group. OWL 2 Web Ontology Language Document Overview. W3C Recommendation, 2009. URL http://www.w3.org/TR/owl-overview/.

[17] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C Working Draft, 2010. URL http://www.w3.org/TR/sparql11-query/.

[18] Patrick Hayes. RDF Semantics. W3C Recommendation, 2004. URL http://www.w3.org/TR/rdf-mt/.

[19] Ivan Herman. Public W3C Questionnaire on RDF Evolution, 2010. URL http://www.w3.org/blog/SW/2010/08/18/public_w3c_questionnaire_on_rdf_evolutio.

[20] Mary Holstege and Asir S. Vedamuthu. W3C XML Schema Definition Language (XSD): Component Designators. W3C Candidate Recommendation, 2010. URL http://www.w3.org/TR/xmlschema-ref/.

[21] Frank Manola and Eric Miller. RDF Primer. W3C Recommendation, 2004. URL http://www.w3.org/TR/rdf-primer/.

[22] Deborah L. McGuinness and Frank van Harmelen. OWL Web Ontology Language Overview. W3C Recommendation, 2004. URL http://www.w3.org/TR/owl-features/.

[23] Boris Motik. A proposal for ISSUE-104 (built-in vocabulary), 2008. URL http://lists.w3.org/Archives/Public/public-owl-wg/2008Jun/0070.html.

[24] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. W3C Recommendation, 2008. URL `http://www.w3.org/TR/rdf-sparql-query/`.

[25] Andy Seaborne. sparql11-query. W3C Working Draft, 2010. URL `http://www.w3.org/TR/sparql11-property-paths/`.

[26] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn. W3C XML Schema Part 1: Structures Second Edition. W3C Recommendation, 2004. URL `http://www.w3.org/TR/xmlschema-1/`.

II. rész

RDF kinyerés

3. fejezet

RDF kinyerő konverziós programok

3.1. Bevezetés

Ebben a fejezetben a szerző által kifejlesztett RDF konverziós eszközök kerülnek bemutatásra. A 3.2. szakasz a kapcsolódó problémakör egy rövid áttekintését adja, majd két olyan saját fejlesztésű eszköz ismertetése következik, amelyek a szerző hozzájárulását jelentik a 3.2.4. szakaszban említésre kerülő RDFizers projekthez. Végül a 3.6. szakasz a szerző az előbbi eszközöket egységes keretbe ágyazó RDF kinyerő keretrendszerét tárgyalja.

3.2. RDF adatforrások

A szemantikus web megvalósulásának előfeltétele az információk RDF-ben rendelkezésre állása. Választ kell adnunk tehát arra a kérdésre, hogy miként juthatnak hozzá az alkalmazások adott erőforrást leíró RDF kijelentésekhez.

3.2.1. Kapcsolódó metaadat erőforrás társítása

Elméletileg tetszőleges erőforráshoz társítható egy annak leírását RDF-ben szolgáltató másik erőforrás. Például HTML és XML dokumentumokhoz a társítás megvalósításához szabványos megoldás létezik. Az RDF gráfok XML szintaxisát (RDF/XML) definiáló [23] szabvány egy alkalmas mechanizmust ad metaadatokat hordozó külső RDF/XML dokumentumok HTML és XML dokumentumokhoz kapcsolásához. A gyakorlat általában statikus RDF/XML dokumentumok használatára korlátozódik. Noha ez egy egyszerű

és kézenfekvő megoldás, meglehetősen rugalmatlan és kényelmetlen, éppen ezért nem is túl népszerű és elterjedt.

3.2.2. RDFa

A W3C RDFa szabványa [21, 22] egy egyszerű és elegáns megoldást ad RDF kijelentések XHTML dokumentumokba beágyazásához. A beágyazás implicit módon, speciális XML attribútumok felhasználásával történik. Egy feldolgozási modell határozza meg a dokumentumból az RDF hármasok kinyerésére szolgáló feldolgozási szabályokat. Az RDFa használata jelenleg az XHTML 1.1 számú verziójához definiált. Ígéretes megoldás, amelyet számos szoftver – például böngésző kiterjesztés és fejlesztőeszköz – támogat. Az RDFa eszközök és felhasználások felsorolását lásd például az RDFa Wiki-ben [14].

3.2.3. GRDDL

Ugyancsak W3C szabvány a GRDDL [25], amely lehetővé teszi RDF hármasok kinyerésére szolgáló transzformációk hozzárendelését XML dokumentumokhoz. URI-k szolgálnak a transzformációk azonosítására, amelyek megvalósítása a gyakorlatban többnyire XSLT stíluslapokkal történik, noha a szabvány nem zárja ki az egyéb megoldásokat (például szkriptek, programok alkalmazását).

3.2.4. RDF kinyerő konverziós eszközök

Az eddig bemutatott megoldások kizárólag weboldalakhoz és XML dokumentumokhoz használhatók, azonban a weben számtalan egyéb fajta erőforrás érhető el. Gondoljunk csak az olyan elterjedten használt bináris állományformátumokra, mint például a PDF vagy JPEG! Ezek közös jellemzője, hogy a legtöbb esetben metaadatokat is hordoznak az állományok.

Sok formátumhoz rendelkezésre állnak olyan konverziós eszközök, melyek metaadatok RDF hármasok formájában történő kinyerésére szolgálnak. A különböző erőforrásokból RDF kijelentéseket kinyerni tudó konverziós eszközökre gyakran használják az **RDFizer** kifejezést, amely eredetileg egy SIMILE[1] alprojekt fedőneve.

RDF kinyerés az erőforrások feldolgozásával valósulhat meg, amelyhez azonban elengedhetetlen a formátum szerkezetének pontos ismerete. Szerencsére sok formátumhoz adottak olyan programkönyvtárak, amelyek prog-

[1] A SIMILE [17] az MIT Computer Science and Artificial Intelligence Laboratory (MIT CSAIL) és a MIT Libraries közös, a nyílt forrás iránt elkötelezett projektje, amelynek keretében több a szemantikus webhez kapcsolódó fejlesztés is folyik.

ramozási nyelvekből teszik lehetővé az állományok manipulálását. Megfelelő támogatás hiányában a feldolgozást saját kezűleg kell elvégezni, amely jelentős munkát igényelhet.

Az RDFizers projekt [15] keretében olyan konverziós eszközök egy heterogén gyűjteménye érhető el, amelyeket a projekt személyzetének tagjai és külső közreműködők fejlesztettek. Nincs semmiféle megkötés a megvalósításra – így például annak programozási nyelvére vagy az eszköz használatának módjára –, az eszközök egyetlen közös jellemzője az, hogy valamilyen fajta erőforrásokból RDF kijelentéseket állítanak elő valamilyen alkalmas formában.

Konverziós eszközök állnak rendelkezésre például BibTeX állományok, JPEG képállományok és email üzenetek feldolgozásához. A konverzió során cél a konvertált adatokról hasznos információkat szolgáltató lehető legtöbb RDF kijelentés előállítása. Strukturált szerkezetű források konvertálása gyakran értelemszerűen végezhető el, egyéb esetekben emberi beavatkozás lehet szükséges.

Míg a fenti eszközök a megvalósítást tekintve nagyon heterogének, léteznek már az egységesítést megcélzó konverziós keretrendszerek is. Például a nyílt forrású Aperture [3] egy olyan Java-ban készült metaadat kinyerő rendszer, amely megfelelő osztályokat és interfészeket definiál az RDF-kinyerés általános megvalósításához, egyben számos állományformátumhoz biztosít implementációkat.

A legnagyobb RDF kinyerő eszközökből álló gyűjteményt jelenleg minden bizonnyal a Virtuoso Universal Server [19] részét alkotó Virtuoso Sponger mondhatja magáénak. A Virtuoso Sponger egy olyan köztes réteg, amely a legkülönfélébb forrásokból képes kapcsolt adatokat szolgáltatni. Jelenleg százötvennél több *cartridge*-nek nevezett RDF kinyerő áll hozzá rendelkezésre. Az RDF kinyerést elsősorban XSLT stíluslapokkal valósítják meg, amelynek egyenes következményeként XML formátumok feldolgozása jöhet szóba.[2]

Az elérhető RDF kinyerő eszközöket számba vevő egy-egy gyűjteményt találunk a W3C által üzemeltetett [1] és [2] wiki oldalakon is.

3.2.5. XMP

Az Extensible Metadata Platform (XMP) [20] az Adobe Systems RDF-alapú metaadat keretrendszere, amely lehetővé teszi metaadatok beágyazását állományokba úgynevezett XMP csomagok formájában. Számos formátumhoz

[2]Például webszolgáltatások XML formátumai. A Sponger számos népszerű webszolgáltatást támogat, köztük például az Amazon, az eBay, a Flickr és a Google webszolgáltatásait.

meghatározza a beágyazás fizikai megvalósításának módját is. Nagyszerűsége abban rejlik, hogy a beágyazás révén a metaadatok együtt utaznak az állománnyal annak átvitele során. A megoldásról részletesebben a 4. fejezetben olvashatunk.

3.3. RDF kinyerés BitTorrent metainfo állományokból

3.3.1. Bevezetés

A BitTorrent napjaink egyik legnépszerűbb P2P fájlcserélő megoldása, amely a világ teljes internetforgalmának számottevő részét generálja.[3] A BitTorrent kifejezés az alapul szolgáló kommunikációs protokollt takarja, egyben egy kliensprogram neve a sok közül. A név sokak számára – nem teljesen alaptalanul – egyet jelent tartalmak illegális terjesztésével, holott sokan használják törvénybe nem ütköző módon saját tartalmak hatékony megosztására. Gyakran alkalmazzák például szabad és nyílt forrású szoftver projektekben a saját szerverek tehermentesítéséhez. Többek között elérhető BitTorrent-en keresztül az OpenOffice.org irodai csomag, a szabad és nyílt forrású operációs rendszerek közül a Debian, a Fedora és az Ubuntu.

A szakasz a szerző által kifejlesztett konverziós programot mutatja be, amely BitTorrent metainfo állományok információtartalmát alakítja RDF-be.

3.3.2. Metainfo állományok

Az állománymegosztás olyan állományok segítségével történik, amelyek a megosztandó állományok metaadatait tartalmazzák, valamint az adatcseréhez szükséges további információkat. Ezeket az állományokat a legtöbben torrent néven ismerik az elterjedten használt állománynév utótag miatt, noha hivatalosan **metainfo állományok**nak nevezik őket.

Az állományformátumot a [4] dokumentum definiálja. A metainfo állományok speciálisan kódolt információkat tartalmaznak, a kódolási eljárás neve **bencoding**. A formátum alig néhány adattípusra és adatszerkezetre épül. Előjeles decimális egészek, bájtsorozatok, listák és asszociatív tömbök állnak rendelkezésre, amelyek mindegyikéhez egy speciális ábrázolás definiált. A kódolás olyan bájtsorozatokkal történik, amelyekben bizonyos bájtokat a

[3] Például egy idei Cisco-tanulmány szerint [5] 2009-ben a teljes fogyasztói internetforgalom 39%-a P2P fájlcserélő hálózatok számlájára volt írható.

megfelelő ASCII karakterként kell tekinteni. Alább részletezzük a biztosított adattípusokat és adatszerkezeteket:

Előjeles decimális egészek A használható decimális számjegyek maximális számára nincs előírás, a gyakorlatban a számok 64-bites előjeles egészekként kezelése a jellemző. Az alábbi felépítésű ASCII karaktersorozat ábrázol minden előjeles decimális egész számot:

- az első karakter a szám elejét jelző `i`
- ha a szám negatív, akkor a második karakter az előjelnek megfelelő `-`
- a szám decimális számjegyei következnek (redundáns 0 számjegyek nem megengedettek)
- az utolsó karakter a szám végét jelző `e`

A fentieknek megfelelően például az 1234 decimális egész számot az `i1234e` ASCII karaktersorozat reprezentálja.

Bájtsorozatok Adott n hosszúságú tetszőleges bájtsorozat ábrázolása az alábbi bájtsorozattal történik:

- a bájtsorozat elejét az előjel nélküli decimális egészként ábrázolt n számjegyeit szolgáltató ASCII karakterek alkotják
- az eredeti bájtsorozat hosszát megadó decimális számjegy karaktereket egy `:` karakter követi
- végül az eredeti bájtsorozat következik

Például a `hello` karakterláncot az `5:hello` ASCII karaktersorozat ábrázolja.

Listák A listák tetszőleges számú olyan elemet tartalmazó sorozatok, amelyek lehetnek decimális egészek, bájtsorozatok, listák és asszociatív tömbök is. A listákat ábrázoló bájtsorozatok felépítése az alábbi:

- az első bájt a lista elejét jelző `l` karakter
- majd az elemeket kódoló bájtsorozatok következnek a megfelelő sorrendben
- az utolsó bájt a lista végét jelző `e` karakter

Ennek megfelelően például a `li13e4:helpe` ASCII karaktersorozat egy olyan kételemű listát ábrázol, amelynek első eleme a 13 decimális egész, második eleme pedig a `help` karakterlánc.

Asszociatív tömbök Az asszociatív tömbök olyan kulcs-érték párokat tartalmazó adatszerkezetek, amelyekben a kulcsok ASCII karakterláncok, a hozzájuk tartozó értékek pedig decimális egészek, bájtsorozatok, listák és asszociatív tömbök is lehetnek. Asszociatív tömbök ábrázolása az alábbi felépítésű bájtsorozatokkal történik:

- az első bájt az asszociatív tömb elejét jelző `d` karakter
- majd a kulcs-érték párokat kódoló bájtsorozatok következnek (a párokban a kulcsot a hozzá tartozó érték követi)
- az utolsó bájt az asszociatív tömb végét jelző `e` karakter

Például a `d3:cat4:meow1:ni42ee` ASCII karaktersorozat egy olyan asszociatív tömböt ábrázol, amelyben a `cat` kulcshoz a `meow` ASCII karakterlánc tartozik, az `n` kulcshoz pedig a 42 decimális egész.

Minden metainfo állomány egy asszociatív tömböt tartalmaz, amelyben a formátum által meghatározott kulcs-érték párok szerepelnek. Minden egyes kulcshoz egyedileg meghatározott a hozzá tartozó érték értelmezése. Például egy kulcshoz tartozó bájtsorozat reprezentálhat bináris adatokat vagy egy adott karakterkódolásban kódolt szöveget is. A lista és az asszociatív tömb olyan rekurzív adatszerkezetek, amelyek bonyolult, rekord-szerű struktúrák felépítését is lehetővé teszik.

A formátum által meghatározott metaadat elemek száma szerencsére nem túl nagy, azonban a protokollt implementáló kliensprogramok használhatnak implementáció-specifikus bővítéseket.

3.3.3. A kinyerés megvalósítása

A szerző kifejlesztett egy olyan a metainfo formátumot implementáló saját szoftvert, amely a metainfo állományok tartalmát RDF-be alakítja. Ehhez kidolgozott egy megfelelő RDF szókészletet, amely a szabványos metaadat elemeket fedi le. A szoftver Java-ban készült, a metainfo állomány tartalmát reprezentáló RDF gráf létrehozásához a Jena Semantic Web Framework [10] keretrendszert használja.

Természetes módon adja magát a metainfo állomány egy RDF gráfra történő leképezése. A konverzió teljesen gépiesen, az adatelemek jelentésének ismerete nélkül is elvégezhető. Az átalakítás során a metainfo konstrukciók megfelelő XML séma és RDF konstrukciókra képeződnek le, a megfeleltetés a 3.1. ábrán látható. Noha az RDF gráffá alakítás lehetne teljesen gépies, a program mégsem így működik.

Metainfo konstrukció	XSD/RDF konstrukció
előjeles decimális egész (szám)	`xsd:long`
előjeles decimális egész (POSIX idő)	`xsd:dateTime`
előjeles decimális egész (kétértékű: 0 vagy 1)	`xsd:boolean`
bájtsorozat (UTF-8 kódolt szöveg)	`xsd:string`
bájtsorozat (bináris adat)	`xsd:base64Binary`
bájtsorozat (URI)	RDF URI
lista	RDF kollekció
asszociatív tömb	üres csomópont a kulcsoknak megfelelő tulajdonságokkal

3.1. ábra. Metainfo konstrukciók leképezése

Mivel az állományban megjelenő decimális egész számok és bájtsorozatok értelmezése az előfordulás helyétől függ, a program a szabványos kulcsokhoz beépítve tartalmazza a megfelelő XML séma céltípusokat.

Másrészt részben eltérő a csak egyetlen állomány megosztását szolgáló és az egyidejűleg több állomány megosztását szolgáló metainfo állományok felépítése. A táblázat alapján történő gépies konverzió a két esetben nem kívánatos módon eltérő, ráadásul mindkét esetben redundáns elemeket is tartalmazó RDF gráfszerkezetet eredményezne. Ezeknek a problémáknak a kiküszöböléséhez a program alkalmas módon átstrukturálja a gráfot, egységes és esztétikusabb felépítést biztosítva.

Az eredmény gráfban kiegészítő információk is elhelyezésre kerülnek:

- A `file-mode` tulajdonság, amelynek értéke `single` vagy `multiple`, azt jelzi, hogy az eredeti metainfo állomány egyetlen állomány vagy több állomány megosztására szolgál. (Ezt csupán a metainfo állomány felépítése tükrözi.)

- A logikai értékű `extensions` tulajdonság azt jelzi, hogy a metainfo állomány használ-e nem szabványos és a program által nem kezelt kiterjesztéseket.

Felhasználói beállításként adható meg, hogy hogyan történjen a metainfo állományban megjelenő, de a program által nem támogatott nem szabványos adatelemek kezelése: választható figyelmen kívül hagyás vagy gépies konverzió. Utóbbi esetben egy alkalmas karakterkódolásban adott szövegként értelmezhető bájtsorozatot a program automatikusan az `xsd:string` típusra képez le, egyébként a bájtsorozatot bináris adatként tekinti és a konverzió céljának az `xsd:base64Binary` típust tekinti. A gépies konverzió esetén a program továbbá a nem szabványos adatelemekhez olyan alkalmasan elnevezett tulajdonságokat használ, amelyek nem az RDF szókészlet névterébe tartoznak, hanem egy az ismeretlen metaadat elemek számára fenntartott névtérbe.

A 3.2. ábra szemlélteti egy metainfo állományra végrehajtott gépies konverzió eredményét, a 3.3. ábra pedig a program által előállított RDF gráfot mutatja.

A metainfo állományokhoz a szerző által kidolgozott RDF szókészletet egy OWL 1 Full webontológia definiálja, amely a `http://purl.org/net/vocabulary/bittorrent.owl` címen található. Az ontológia azért van OWL Full-ban, mert a metainfo listák ábrázolása RDF kollekciókkal történik. (Sajnos OWL DL-ben az RDF kollekciók csupán magának az ontológiának az ábrázolásához állnak rendelkezésre.)

3.3.4. A program használata

A program fordításához és használatához Java fejlesztői környezet (Java SE 6 az ajánlott) valamint az Apache Maven 3 [11] telepítése szükséges. A program két további függőséget igényel: az RDF gráf létrehozásához a Jena Semantic Web Framework [10] keretrendszert, parancssori argumentumok feldolgozásához pedig a `Commons CLI` [6] programkönyvtárat használja (mindkettő szabad és nyílt forrású). Futtatáshoz egy parancssoros interfész áll rendelkezésre, amelynek lehetőségeit alább láthatjuk:

```
usage: java hu.unideb.inf.rdfizers.bittorrent.Main [options]
 -f,--file <file>           read input from the file specified
 -h,--help                  display this help and exit
 -o,--output <file>         write output to the file specified instead
                            of standard output
 -l,--language <language>   write output in the language specified
                            (N-TRIPLES, TURTLE, RDF/XML,
                            RDF/XML-ABBREV, default: RDF/XML)
 -ie,--ignore-exts          ignore unknown implementation specific
                            extensions
 -u,--url <url>             read input from the URL specified
```

```
@prefix :        <http://purl.org/net/vocabulary/bittorrent.owl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

<http://torrent.fedoraproject.org/torrents/Fedora-13-x86_64-DVD.torrent>
      a       :MetainfoFile ;
      :announce "http://torrent.fedoraproject.org:6969/announce"^^xsd:string ;
      :creation-date "1274736010"^^xsd:long ;
      :info   [
                  :files  ([
                             a       :File ;
                             :length "1612"^^xsd:long ;
                             :path   ("Fedora-13-x86_64-CHECKSUM"^^xsd:string)
                           ]
                           [
                             a       :File ;
                             :length "3630045184"^^xsd:long ;
                             :path   ("Fedora-13-x86_64-DVD.iso"^^xsd:string)
                           ]) ;
                  :name "Fedora-13-x86_64-DVD"^^xsd:string .
                  :piece-length "262144"^^xsd:long ;
                  :pieces "XrVkrmAeV...RsctQ5+9aK"^^xsd:base64Binary
              ] .
```

3.2. ábra. Gépiesen RDF-be konvertált metainfo állomány Turtle szintaxisban ábrázolva

```
@prefix :        <http://purl.org/net/vocabulary/bittorrent.owl#> .
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .

<http://torrent.fedoraproject.org/torrents/Fedora-13-x86_64-DVD.torrent>
      a       :MetainfoFile ;
      :announce <http://torrent.fedoraproject.org:6969/announce> ;
      :creation-date "2010-05-24T21:20:10Z"^^xsd:dateTime ;
      :file-mode "multiple"^^xsd:string ;
      :files  ([
                  a       :File ;
                  :length "1612"^^xsd:long ;
                  :path   ("Fedora-13-x86_64-CHECKSUM"^^xsd:string)
                ]
                [
                  a       :File ;
                  :length "3630045184"^^xsd:long ;
                  :path   ("Fedora-13-x86_64-DVD.iso"^^xsd:string)
                ]) ;
      :piece-length "262144"^^xsd:long ;
      :pieces "XrVkrmAeV...RsctQ5+9aK"^^xsd:base64Binary ;
      :target-directory "Fedora-13-x86_64-DVD"^^xsd:string ;
      :extensions "false"^^xsd:boolean .
```

3.3. ábra. RDF-be konvertált metainfo állomány Turtle szintaxisban ábrázolva (a program által adott eredmény)

A megvalósítás biztosít továbbá olyan osztályokat, amelyek lehetővé teszik az RDF kinyerő beillesztését a fejezet utolsó szakaszában tárgyalt keretrendszerbe. A program elérhető a szerző honlapján [13].

3.4. RDF kinyerés RPM csomagokból

3.5. Bevezetés

Számos Linux-disztribúció csomagkezelése alapul az RPM Package Manager (RPM) csomagkezelő programon, amelyeket összefoglaló néven RPM-alapú disztribúcióknak neveznek. A csomagkezelő által használt csomagok állományformátumát szintén RPM-nek nevezik. A szakasz a szerző egy saját fejlesztésű RDF kinyerő programját mutatja be, amely RPM csomagokat dolgoz fel.

A csomagkezelés kapcsán az 5.3.1.2. alszakaszban olvashatunk az RPM-alapú rendszerekről, további vonatkozó történeti megjegyzéseket is tartalmaz az A. függelék.

3.5.1. A kinyerés megvalósítása

Az RPM csomagok speciális szerkezetű bináris állományok, amelyek feldolgozása a formátumot megvalósító programokat igényel. Referencia implementációként az `rpmlib` nevű C programkönyvtár tekinthető [27]. A szerző RDF kinyerő programja Java programozási nyelven készült. A metaadatok feldolgozásához szükséges szinten saját maga valósítja meg a formátum kezelését, ehhez nem használ külső programkönyvtárakat.[4]

Az RPM állományok felépítését az A. függelék tárgyalja részletesen. A program a metaadatokat tartalmazó szignatúra és fejléc rész leképezését valósítja meg egy RDF gráfra. A metaadatok tárolása egy fejléc struktúrának nevezett adatszerkezetben történik, amely lehetővé teszi metaadat címkékhez értékként tipizált adatok hozzárendelését.

A 3.4. ábrán látható az RPM adattípusok megfeleltetése XML séma és RDF konstrukcióknak. (A NULL és INT64 típusokat nem használja a formátum.) Az állományformátum lehetővé teszi CHAR, INT8, INT16 és INT32 típusú tömbök használatát, amelyeket a program RDF konténerekkel ábrázol.

[4]Ma már létezik Java programkönyvtár RPM csomagok kezeléséhez, egy ilyen szabad és nyílt forrású megoldás például a `Redline` [16]. Amikor a szerző programja eredetileg készült, akkor még nem állt rendelkezésre hasonló eszköz.

RPM adattípus	XSD/RDF konstrukció
NULL	–
CHAR	`xsd:string`
INT8	`xsd:byte`
INT16	`xsd:short`
INT32	`xsd:int`/`xsd:dateTime`
INT64	–
STRING	`xsd:string`
BIN	`xsd:base64Binary`
STRING_ARRAY	`rdf:Seq` (`xsd:string`)
I18NSTRING	`rdf:Alt` (`xsd:string`)

3.4. ábra. RPM adattípusok leképezése

Vegyük észre, hogy a fejléc struktúra kézenfekvő módon képezhető le automatikusan egy RDF gráfra: ábrázolja egy olyan erőforrás a csomagot, amely a gráfot alkotó kijelentések alanyaként jelenik meg, a kijelentések tartalmazzák állítmányként a címkéket, tárgyként pedig ezek értékeit. (Tömbök kezeléséhez szükségesek továbbá konténereket ábrázoló üres csomópontok.)

A szemantikát figyelmen kívül hagyó gépies konverzió sajnos az igényeknek nem megfelelő RDF gráfot eredményez:

- A címkék típusának ismerete sok esetben nem elégséges az adatok értelmezéshez. Például egy INT32 típusú előjeles egész bizonyos címkék esetében az `xsd:dateTime` típussal ábrázolható időbélyeget jelent. Még a címkék neve sem áll rendelkezésre, ezek azonosítása az állományokban egész számokkal történik. (Például a fejlécben 1004 a csomag licencét szolgáltató LICENSE címkét jelenti.)

- A címkék logikailag összetartozhatnak. Például a csomagban tartalmazott állományok metaadatait tucatnyinál több olyan címke szolgáltatja, amelynek értékeként az állományok számának megfelelő elemszámú tömbök jelennek meg. A megfelelő tömb i-edik eleme tartalmazza az i-edik állomány utolsó módosításának idejét, egy másik tömb i-edik eleme ugyanannak az állománynak a méretét, és így tovább.

- Vannak olyan kódolt információkat tartalmazó összetartozó címkék, amelyeket nem lehet önmagukban értelmezni. Például a csomagban tartalmazott állományok elérési útvonalait három tömb felhasználásával kell dekódolni.

A szerző programja az adatok szemantikáját tükröző esztétikus felépítésű RDF gráfokat állít elő. Ehhez beépítve tartalmazza mindazt a tudást, amely a fejléc struktúra megfelelő értelmezéséhez szükséges. A konverzió során néhány címkét figyelmen kívül hagy.[5]

A fejléc struktúrában tartalmazott metaadatok az alábbi főbb csoportokba sorolhatók:

1. A csomagot jellemző skalár értékek: például a csomag neve, verziószáma, rövid (egysoros) és hosszú leírása.

2. Függőségekre vonatkozó metaadatok.

3. A csomagban tartalmazott állományok metaadatai.

4. A csomag változásait dokumentáló úgynevezett **changelog** bejegyzések.

A program kimenetként az RDF/XML szintaxisban ábrázolja a konverzió eredményeként előálló gráfot. Az RDF/XML dokumentumot a Java platform részét alkotó Streaming API for XML (StAX) [18] segítségével hozza létre. A csomagok leírásához egy olyan RDF szókészletet használ, amelyet a `http://purl.org/net/vocabulary/rpm` címen elérhető dokumentum definiál, amelyben az osztályok és tulajdonságok leírása az RDF Schema szókészlettel történik.

Példával szemléltetjük, hogy a program hogy valósítja meg a csomag metaadatok leképezését egy RDF gráfra. Mivel az eredményül kapott állomány a kísérleti nyúlként választott `kernel` csomag esetében hatalmas méretű[6], több ábrát használunk, amelyek mindegyike a gráf egy jellegzetes részét mutatja. A 3.5. ábrán az RDF/XML állomány eleje látható a skalár értékű tulajdonságokkal. Terjedelmi okból rövidítve ábrázoljuk az `xsd:base64Binary` típusú literálokat és a csomagot azonosító URI-t. A 3.5. ábra a függőségek ábrázolását mutatja, a 3.7. ábra a csomagban tartalmazott állományokról rendelkezésre álló információkat, végül a 3.8. ábra a csomag változásainak dokumentálását.

A changelog bejegyzések konvertálásához karakterlánc feldolgozás szükséges, mivel a módosítást végző személy nevét, email címét és a vonatkozó verziószámot (utóbbi opcionális) egyetlen karakterlánc tartalmazza. A bejegyzés létrehozójának ábrázolásához a FOAF [24] RDF szókészlet használt.

[5]Olyan technikai jellegű metaadatok nem kerülnek bele a gráfba, mint például a fejléc struktúrára számolt ellenőrzőösszeg vagy a fejléc mérete.

[6]A megjelenítéshez esztétikusra formázott XML dokumentum közel negyvenezer sorból áll és másfél megabájt méretű!

```
<?xml version="1.0" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:rpm="http://purl.org/net/vocabulary/rpm#" xmlns:dc="http://purl.org/dc/elements/1.1/"
  xmlns:dcterms="http://purl.org/dc/terms/" xmlns:foaf="http://xmlns.com/foaf/0.1/">
  <rpm:Package
    rdf:about="http://fedora.inode.at/ ... /Fedora/x86_64/os/Packages/kernel-2.6.33.3-85.fc13.x86_64.rpm">
    <rpm:rsa rdf:datatype="http://www.w3.org/2001/XMLSchemabase64Binary">iQIVAwUAS ... Bt9qJDYns=</rpm:rsa>
    <rpm:sha1>89fc51fb3b1fe39e162c4a4aa35645b0a1e2a3b0</rpm:sha1>
    <rpm:pgp rdf:datatype="http://www.w3.org/2001/XMLSchemabase64Binary">iQIVAwUAS ... lNHVGYtQuU=</rpm:pgp>
    <rpm:md5 rdf:datatype="http://www.w3.org/2001/XMLSchemabase64Binary">f/MipdHUdyEpGAIIEjGgWQ==</rpm:md5>
    <rpm:payloadsize>87791624</rpm:payloadsize>
    <rpm:url rdf:resource="http://www.kernel.org/"/>
    <rpm:buildtime rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
      >2010-05-06T20:35:36.000+02:00</rpm:buildtime>
    <rpm:name>kernel</rpm:name>
    <rpm:version>2.6.33.3</rpm:version>
    <rpm:release>85.fc13</rpm:release>
    <rpm:summary>The Linux kernel</rpm:summary>
    <rpm:description>The kernel package contains the Linux kernel (vmlinuz), the core of any Linux
      operating system. The kernel handles the basic functions of the operating system: memory
      allocation, process allocation, device input and output, etc.</rpm:description>
    <rpm:buildhost>x86-02.phx2.fedoraproject.org</rpm:buildhost>
    <rpm:size>108294047</rpm:size>
    <rpm:distribution>Fedora Project</rpm:distribution>
    <rpm:vendor>Fedora Project</rpm:vendor>
    <rpm:license>GPLv2</rpm:license>
    <rpm:packager>Fedora Project</rpm:packager>
    <rpm:group>System Environment/Kernel</rpm:group>
    <rpm:os>linux</rpm:os>
    <rpm:arch>x86_64</rpm:arch>
    <rpm:sourcerpm>kernel-2.6.33.3-85.fc13.src.rpm</rpm:sourcerpm>
    <rpm:rpmversion>4.8.0</rpm:rpmversion>
    <rpm:optflags>-O2 -g -pipe -Wall -Wp,-D_FORTIFY_SOURCE=2 -fexceptions -fstack-protector
      --param=ssp-buffer-size=4 -m64 -mtune=generic</rpm:optflags>
    <rpm:payloadformat>cpio</rpm:payloadformat>
    <rpm:payloadcompressor>xz</rpm:payloadcompressor>
    <rpm:platform>x86_64-redhat-linux-gnu</rpm:platform>
    ...
  </rpm:Package>
</rdf:RDF>s
```

3.5. ábra. RPM csomagból kinyert metaadatok

```
...
<rpm:depends>
  <rpm:Capability>
    <rpm:name>rpmlib(VersionedDependencies)</rpm:name>
    <rpm:minVersionInclusive>3.0.3-1</rpm:minVersionInclusive>
  </rpm:Capability>
</rpm:depends>
<rpm:depends>
  <rpm:Capability>
    <rpm:name>fileutils</rpm:name>
  </rpm:Capability>
</rpm:depends>
<rpm:depends>
  <rpm:Capability>
    <rpm:name>initscripts</rpm:name>
    <rpm:maxVersionInclusive>8.11.1-1</rpm:maxVersionInclusive>
  </rpm:Capability>
</rpm:depends>
...
<rpm:provides>
  <rpm:Capability>
    <rpm:name>kernel</rpm:name>
    <rpm:version>2.6.33.3-85.fc13</rpm:version>
  </rpm:Capability>
</rpm:provides>
<rpm:provides>
  <rpm:Capability>
    <rpm:name>kernel-x86_64</rpm:name>
    <rpm:version>2.6.33.3-85.fc13</rpm:version>
  </rpm:Capability>
</rpm:provides>
<rpm:provides>
<rpm:provides>
  <rpm:Capability>
    <rpm:name>linux-gate.so.1</rpm:name>
  </rpm:Capability>
</rpm:provides>
...
<rpm:conflicts>
  <rpm:Capability>
    <rpm:name>kernel-smp</rpm:name>
  </rpm:Capability>
</rpm:conflicts>
...
```

3.6. ábra. RPM csomagból kinyert metaadatok (függőségek)

```
...
<rpm:files>
  <rdf:Bag>
    <rdf:li>
      <rpm:File>
        <rpm:name>/boot/vmlinuz-2.6.33.3-85.fc13.x86_64</rpm:name>
        <rpm:size>3510976</rpm:size>
        <rpm:username>root</rpm:username>
        <rpm:groupname>root</rpm:groupname>
        <rpm:lastmodified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2010-05-06T20:23:57.000+02:00</rpm:lastmodified>
      </rpm:File>
    </rdf:li>
    <rdf:li>
      <rpm:File>
        <rpm:name>/etc/ld.so.conf.d/kernel-2.6.33.3-85.fc13.x86_64.conf</rpm:name>
        <rpm:size>324</rpm:size>
        <rpm:username>root</rpm:username>
        <rpm:groupname>root</rpm:groupname>
        <rpm:lastmodified rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2010-05-06T20:25:10.000+02:00</rpm:lastmodified>
      </rpm:File>
    </rdf:li>
    ...
  </rdf:Bag>
</rpm:files>
...
```

3.7. ábra. RPM csomagból kinyert metaadatok (tartalmazott állományok)

A megvalósítás során problémát jelentett, hogy a formátum nem megfelelően dokumentált. A Fedora Dokumentációs Projekt [8] keretében történik a témában alapvetőnek számító *Fedora RPM Guide* [26] című könyv fejlesztése. Ennek ellenére a szemléltető ábrák létrehozásához használt Fedora csomagok a könyvben nem dokumentált címkéket használnak. A könyvből az sem derül ki, hogy mi a különbség a STRING_ARRAY és I18NSTRING típusok között. Az `rpmlib` programkönyvtár közel negyvenezer programsort tartalmazó forráskódjában járhat utána az érdeklődő olvasó annak, hogy a két típus kezelése jelenleg azonos módon történik.[7]

3.5.2. A program használata

A program fordításához és használatához Java fejlesztői környezet (Java SE 6 az ajánlott) valamint az Apache Maven 3 telepítése szükséges. A program két további függőséget igényel: Base64 kódoláshoz a `Commons Codec`

[7]A I18NSTRING típust a szerző elképzelése szerint valószínűleg lokalizált karakterláncok kezelésére szánták eredetileg. Azaz olyan tömböt valósíthatna meg, amely feltevés szerint ugyanannak a szövegnek a különböző nyelvű változatait tartalmazza elemekként. Ezt a program működése úgy tükrözi, hogy az egynél több elemszámú I18NSTRING tömbök ábrázolásához az Alt RDF konténert használja.

```
...
<rpm:changelog>
  <rdf:Seq>
    ...
    <rdf:li>
      <rpm:ChangeLogEntry>
        <dc:creator>
          <foaf:Person>
            <foaf:name>Kyle McMartin</foaf:name>
            <foaf:mbox rdf:resource="mailto:kyle@redhat.com"/>
          </foaf:Person>
        </dc:creator>
        <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2010-04-30T14:00:00.000+02:00</dcterms:created>
        <rdfs:comment>- add-appleir-driver.patch: update from hadess, split out some other
          patches. - git-bluetooth.patch: and put them in git-bluetooth, along with other
          fixes.</rdfs:comment>
      </rpm:ChangeLogEntry>
    </rdf:li>
    <rdf:li>
      <rpm:ChangeLogEntry>
        <dc:creator>
          <foaf:Person>
            <foaf:name>Adam Jackson</foaf:name>
            <foaf:mbox rdf:resource="mailto:ajax@redhat.com"/>
          </foaf:Person>
        </dc:creator>
        <dcterms:created rdf:datatype="http://www.w3.org/2001/XMLSchema#dateTime"
          >2010-04-29T14:00:00.000+02:00</dcterms:created>
        <rdfs:comment>- drm-intel-sdvo-fix-2.patch: Require that the A/D bit of EDID match the
          A/D-ness of the connector. (#584229)</rdfs:comment>
      </rpm:ChangeLogEntry>
    </rdf:li>
    ...
  </rdf:Seq>
</rpm:changelog>
...
```

3.8. ábra. RPM csomagból kinyert metaadatok (változások dokumentálása)

[7], parancssori argumentumok feldolgozásához pedig a `Commons CLI` [6] programkönyvtárat használja (mindkettő szabad és nyílt forrású). Futtatáshoz egy parancssoros interfész áll rendelkezésre, amelynek lehetőségeit alább láthatjuk:

```
usage: java hu.unideb.inf.rdfizers.rpm.Main [options]
 -f,--file <file>       read input from the file specified
 -h,--help              display this help and exit
 -o,--output <file>     write output to the file specified instead of
                        standard output
 -oc,--omit-changelog   omit changelog
 -od,--omit-deps        omit dependencies
 -of,--omit-files       omit files
 -u,--url <url>         read input from the URL specified
```

Opciók segítségével előírható, hogy bizonyos fajta metaadatok legyenek a feldolgozás során figyelmen kívül hagyva.

A megvalósítás biztosít továbbá olyan osztályokat, amelyek lehetővé teszik az RDF kinyerő beillesztését a következő szakaszban tárgyalt keretrendszerbe. A teljes program elérhető a szerző honlapján [13].

3.6. Saját RDF kinyerő keretrendszer megvalósítása

3.6.1. Bevezetés

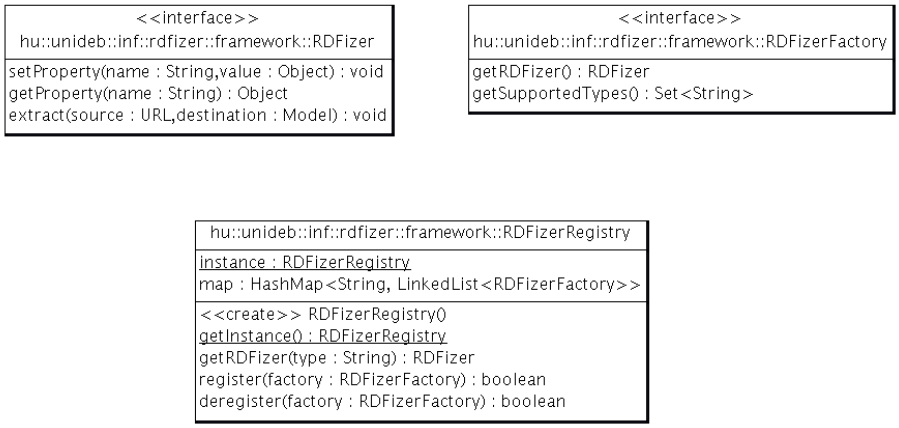
Ebben a szakaszban egy olyan Java keretrendszer kerül bemutatásra, amelyet a szerző a különböző RDF kinyerő eszközök használatának egységesítéséhez dolgozott ki. A keretrendszer a szemantikus web alkalmazások Java platformra fejlesztéséhez széles körben használt Jena Semantic Web Framework rendszerhez illeszkedik, amelytől mindössze egyetlen, az RDF gráfok modellezésére szolgáló interfészt vesz át.

3.6.2. A keretrendszer bemutatása

A 3.9. ábrán láthatjuk a keretrendszer alapjául szolgáló osztályokat és interfészeket.

A keretrendszer olyan kinyerőket kezel, amelyek URI segítségével azonosítható és elérhető erőforrásokat képesek feldolgozni.

Az RDFizer interfész. Az RDF kinyerő osztályok az `RDFizer` interfészt kell hogy megvalósítsák. A célkitűzések között szerepelt, hogy az interfész



3.9. ábra. Az RDF kinyerő keretrendszer alapját jelentő osztályok és interfészek

mögé könnyen elrejthető legyen bármely létező RDF kinyerő eszköz, ezért az a lehető legegyszerűbb, mindössze egyetlen metódust ír elő. Az `extract()` metódus a feldolgozandó erőforrás URI-ját kapja első paraméterként, második paramétere pedig egy olyan RDF gráfot reprezentáló objektum, amelyhez a kinyerés során előállított RDF kijelentéseket hozzá kell adni.[8]

A URI kétféle módon is felhasználásra kerül:

- Alanyként jelenik meg a kinyerés során létrehozott és az erőforrást leíró RDF kijelentésekben.
- A feldolgozás során általa történik az erőforrás elérése.

Kizárólag olyan URI-k használata jöhet szóba, amelyekhez rendelkezésre áll megfelelő protokollkezelő az erőforrás eléréséhez. A Java platform ezt a `http`, `https`, `ftp` és `file` URI sémákhoz biztosítja.[9] Lokális állományok az állományrendszerben azonosításához a `file` URI sémát kell használni a kinyerőknek átadandó URI-kban.

Az RDFizerFactory interfész. Az `RDFizer` interfészt megvalósító objektumok létrehozása az *abstract factory* tervezési minta [28] alapján történik.

[8] A második paraméter típusaként jelenik meg a Jena Semantic Web Framework `com.hp.hpl.jena.rdf.model.Model` interfésze, amely egy RDF gráfot reprezentál.

[9] Lásd a `java.net.URL` osztály API dokumentációját [9].

Az `RDFizerFactory` interfész `getRDFizer()` metódusa szolgál az `RDFizer` objektumok létrehozására. Minden `RDFizerFactory` példánytól adott típusú erőforrások feldolgozására képes kinyerőkhöz lehet így jutni, a támogatott MIME-tartalomtípusokat [12] a `getSupportedTypes()` metódus adja vissza egy halmazban. Implementálható úgy a `getRDFizer()` metódus, hogy minden egyes hívása során egy új kinyerő objektum jön létre, de takarékossági megfontolásokból akár ugyanaz a példány is visszaadható a hívásokban.

Az RDFizerRegistry osztály. A különböző `RDFizerFactory` megvalósítások kezeléséhez a keretrendszer úgynevezett *service provider* megoldást használ, hasonlóan például a JDBC és JNDI alkalmazói programozói interfészekhez. Az `RDFizerRegistry` osztály adminisztrálja a rendelkezésre álló `RDFizerFactory` példányokat. A nyilvántartás alapján képes adott MIME-tartalomtípushoz a megfelelő példány kiválasztására.

Az `RDFizerRegistry` osztály nem példányosítható, egyetlen példánya a `getInstance()` metódussal kapható meg. Az `RDFizerFactory` példányok a `register()` metódussal vehetők nyilvántartásba. Minden MIME-tartalomtípust tetszőleges sok kinyerő kezelhet, ezért a bejegyzések tárolása egy olyan kulcs-érték párokat tartalmazó adatszerkezetben történik, amely minden tartalomtípushoz a megfelelő `RDFizerFactory` példányok listáját rendeli hozzá (a listában a példányok a bejegyzés sorrendjében követik egymást).

A `getRDFizerFactory()` metódus az adott MIME-tartalomtípusú erőforrások feldolgozására képes kinyerőket létrehozó `RDFizerFactory` példányok közül a legutóbbit adja vissza. A `getRDFizerFactories()` értelemszerűen a típushoz rendelkezésre álló valamennyi `RDFizerFactory` példányt szolgáltatja.

Végül a `deregister()` metódus feladata, hogy a paraméterként adott bejegyzett `RDFizerFactory` példányt törje a nyilvántartásból.

A regisztráció automatikusan is elvégezhető egy olyan XML konfigurációs állománnyal, amelynek feldolgozása az osztálybetöltés során történik. Az állományban az `RDFizerFactory` interfészt megvalósító osztályok minősített nevét kell megadni 3.10. ábrán látható módon. A feldolgozó valamennyi megnevezett osztályt az alapértelmezett konstruktorral példányosítja, valamint regisztrálja az eredményül kapott objektumokat.

3.6.3. Hasonló létező rendszerek

Időközben más RDF kinyerő keretrendszerek is születtek. Egy a bemutatott megoldáshoz hasonló megvalósítású részrendszert tartalmaz például az Aperture [3]. A hasonlóság természetes, hiszen nyilvánvalóként kínálja magát *service provider* felépítés.

```
<?xml version="1.0"?>
<!DOCTYPE configuration [
  <!ELEMENT configuration (factory*)>
  <!ELEMENT factory EMPTY>
  <!ATTLIST factory class NMTOKEN #REQUIRED>
]>
<configuration>
  <factory class="hu.unideb.inf.rdfizer.bittorrent.MetainfoRDFizerFactory"/>
  <factory class="hu.unideb.inf.rdfizer.rpm.RPMRDFizerFactory"/>
  ...
</configuration>
```

3.10. ábra. XML konfigurációs állomány a keretrendszer konfigurálásához

A bemutatott saját keretrendszer méreteiben nem mérhető az Aperture rendszerhez, amely a szerző munkájával ellentétben egy nagyméretű közösségi projekt, és amelynek sok felhasználása van az iparban is. Főleg az elérhető kinyerők száma terén szembetűnő az Aperture fölénye.

Irodalomjegyzék

[1] ConverterToRdf – ESW Wiki. URL http://esw.w3.org/ConverterToRdf.

[2] W3C Semantic Web Standards Wiki. URL http://www.w3.org/2001/sw/wiki/Main_Page.

[3] Aperture. URL http://aperture.semanticdesktop.org/.

[4] Bittorrent Protocol Specification v1.0. URL http://wiki.theory.org/BitTorrentSpecification.

[5] Cisco Visual Networking Index: Forecast and Methodology, 2009–2014. White Paper, 2010. URL http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf.

[6] Commons CLI. URL http://commons.apache.org/cli/.

[7] Commons Codec. URL http://commons.apache.org/codec/.

[8] Fedora Documentation Project. URL http://fedoraproject.org/wiki/DocsProject.

[9] Java SE 6 API Documentation. URL http://download.oracle.com/javase/6/docs/.

[10] Jena Semantic Web Framework. URL http://jena.sourceforge.net/.

[11] Apache Maven. URL http://maven.apache.org/.

[12] MIME Media Types. URL http://www.iana.org/assignments/media-types/.

[13] RDFizers developed by Peter Jeszenszky. URL http://www.inf.unideb.hu/~jeszy/rdfizers/.

[14] RDFa Wiki. URL http://rdfa.info/wiki/RDFa_Wiki.

[15] RDFizers. URL http://simile.mit.edu/wiki/RDFizers.

[16] Redline. URL http://redline-rpm.org/.

[17] SIMILE Project. URL http://simile.mit.edu/.

[18] JSR-173 Specification: Streaming API For XML. URL http://jcp.org/en/jsr/detail?id=173.

[19] Virtuoso Universal Server.

[20] Adobe XMP: Adding intelligence to media. URL http://www.adobe.com/products/xmp/.

[21] Ben Adida and Mark Birbeck. RDFa Primer. W3C Recommendation, 2008. URL http://www.w3.org/TR/xhtml-rdfa-primer/.

[22] Ben Adida, Mark Birbeck, Shane McCarron, and Steven Pemberton. RDFa in XHTML: Syntax and Processing. W3C Recommendation, 2008. URL http://www.w3.org/TR/rdfa-syntax/.

[23] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, 2004. URL http://www.w3.org/TR/rdf-syntax-grammar/.

[24] Dan Brickley and Libby Miller. FOAF Vocabulary Specification, 2010. URL http://xmlns.com/foaf/spec/. version 0.97.

[25] Dan Connolly. Gleaning Resource Descriptions from Dialects of Languages (GRDDL). W3C Recommendation, 2007. URL http://www.w3.org/TR/grddl/.

[26] Eric Foster-Johnson. RPM Guide, 2005. URL http://rpm5.org/docs/rpm-guide.pdf.

[27] Eric Foster-Johnson, Stuart Ellis, and Ben Cotton. RPM Guide, 2010. URL http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/.

[28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 978-0-201-63361-0.

4. fejezet

XMP kinyerő böngészőfunkció

4.1. Bevezetés

Ebben a fejezetben a szerző egy olyan fejlesztése kerül bemutatásra, amely egy sokáig egyedülálló böngészőfunkciót valósított meg, lehetővé téve a Firefox böngészőprogramban XMP metaadatok kinyerését és böngészését.

4.2. XMP

Az Extensible Metadata Platform (XMP) [7] az Adobe Systems RDF-alapú metaadat keretrendszere erőforrások leírásához. Erőforrásként tekinthető egy állomány, vagy annak egy olyan része, amely egy feldolgozó alkalmazás számára jelentéssel bírhat, és amely a formátum szempontjából az állományszerkezet egy logikai komponense. Az XMP egy olyan adatmodellt definiál, amelynek ábrázolásához az RDF XML szintaxisának (RDF/XML) [13] egy részhalmazát használja [9].

Olyan szabványos metaadat szókészleteket biztosít továbbá, amelyeket a legkülönfélébb alkalmazások használhatnak erőforrások – például digitális képek, hang- és videó állományok – leírására [10].

Kulcsfontosságú jellemzője, hogy lehetővé teszi metaadatok beágyazását állományokba úgynevezett XMP csomagok formájában. Számos elterjedt formátumhoz meghatározza a beágyazás fizikai megvalósítását is [11]. A támogatott formátumok között vannak képformátumok (például JPEG és PNG), dinamikus média formátumok (például AVI, MP3, MPEG-2, MPEG-4, WAV) és dokumentumformátumok is (például PDF, PostScript).

Előnyeit az alábbiakban foglaljuk össze:

- Szabványos és állományformátumtól független módját adja digitális képek és egyéb erőforrások metaadatokkal annotálásának.

```
<?xpacket begin="" id="W5M0MpCehiHzreSzNTczkc9d"?>
<x:xmpmeta xmlns:x="adobe:ns:meta/">
  <rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <rdf:Description rdf:about=""
        xmlns:dc="http://purl.org/dc/elements/1.1/">
      <dc:format>application/pdf</dc:format>
      <dc:title>
        <rdf:Alt>
          <rdf:li xml:lang="x-default">Test File</rdf:li>
        </rdf:Alt>
      </dc:title>
    </rdf:Description>
    <rdf:Description rdf:about=""
        xmlns:xap="http://ns.adobe.com/xap/1.0/">
      <xap:CreateDate>2008-12-10T10:00:00Z</xap:CreateDate>
      <xap:CreatorTool>
        pdfeTeX 3.141592-1.21a-2.2 (Web2C 7.5.4)
      </xap:CreatorTool>
    </rdf:Description>
    ...
  </rdf:RDF>
</x:xmpmeta>
<?xpacket end="r"?>
```

4.1. ábra. XMP csomag

- Átvitel során a metaadatok a beágyazó erőforrással együtt utaznak, így nem veszhetnek el útközben.
- Az egységesen ábrázolt metaadatok olyan alkalmazások számára is elérhetők, amelyek nem feltétlenül ismerik a beágyazó erőforrás formátumát.
- Új dimenziókat nyit a digitális fotózásban és a képszerkesztő alkalmazások számára
- Ha széles körben támogatott lesz és használata elterjed a weben, hatékonyan kiaknázható metaadatforrásokat fog biztosítani szemantikus web alkalmazások számára.

Mindez mit sem érne megfelelő támogatást biztosító alkalmazások nélkül. Az XMP az Adobe számára stratégiai fontosságú, gyakorlatilag valamennyi terméke (például az Adobe InDesign, Adobe Photoshop és Adobe Reader) támogatja. C++ és Java projektekhez pedig nyílt forrású szoftverként érhető el az XMP Toolkit SDK [8] fejlesztői könyvtár.

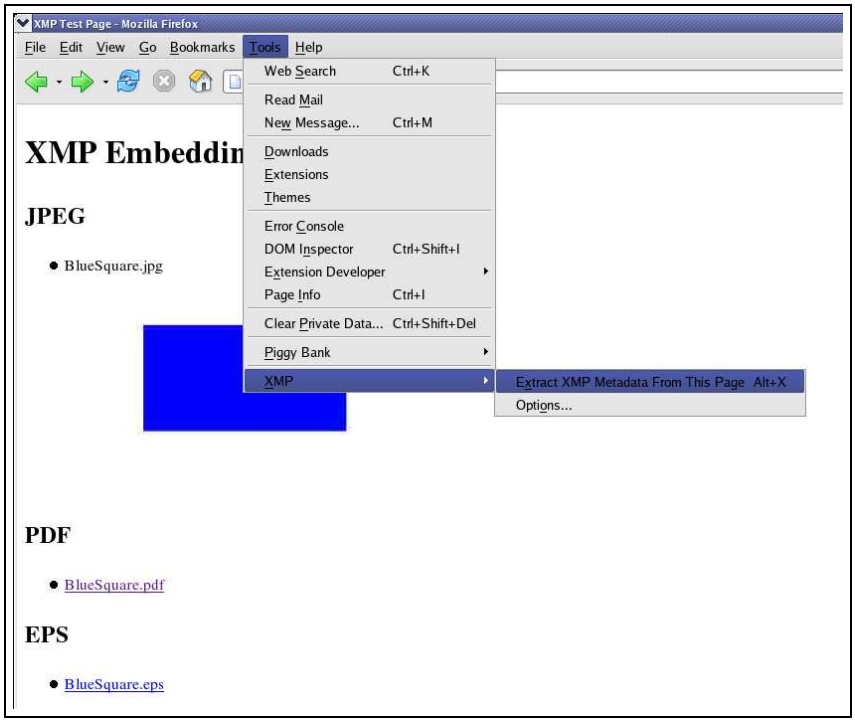
4.3. Piggy Bank

A új böngészőfunkció a szabad és nyílt forrású Piggy Bank [5] böngésző kiterjesztésen alapul. A Piggy Bank a szemantikus web böngészők egyik úttörője. Szemantikus web technológiákat és olyan előremutató megoldásokat alkalmaz, amelyek a böngészésnek egy újfajta élményét adják, és amelyek előrevetítik azt, hogy mit várhatunk az eljövendő szemantikus webtől.

Segítségével a weboldalakból automatikusan nyerhetők ki és tárolhatók el információk, amelyek később rugalmasan kereshetők és szűrhetők. Többféle újszerű módon is lehetővé teszi a kinyert információk megjelenítését, például képes keresési találatokat térképen elhelyezni, események időbeliségét idővonalon ábrázolni, az adatok szerkezetét szemléltetni.

4.4. Az új böngészőfunkció

A funkció használatával egy weboldalról elérhető erőforrásokból lehet kinyerni az azokba beágyazott XMP metaadatokat. A művelet képekre vagy az oldalon hiperhivatkozások célpontjaként megadott erőforrásokra értelmezett. A metaadatok kinyerhetők állományonként, de lehetőség van valamennyi erőforrás egy menetben feldolgoztatására is. Kinyerés után a metaadatok a továbbiakban a Piggy Bank segítségével manipulálhatók.



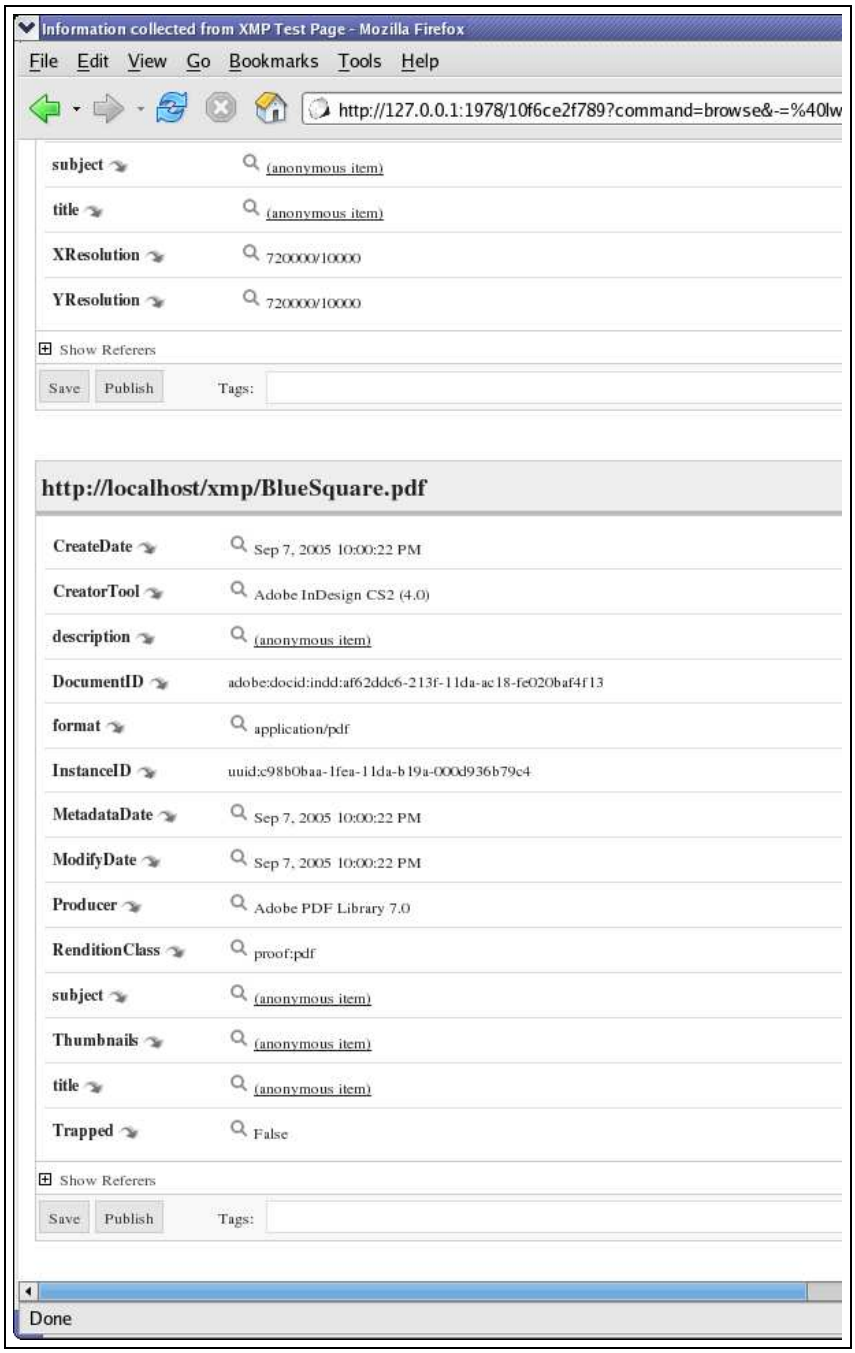
4.2. ábra. Az oldalról elérhető erőforrások XMP metaadatainak kinyerése

A 4.2. és a 4.3. ábrákon figyelhető meg a funkció használata. Előbbin egy olyan oldal böngészése történik, amelyről XMP csomagokat beágyazó erőforrások érhetők el, valamennyire végrehajtható a kinyerés az aktuálisan kiválasztott Tools → XMP → Extract XMP Metadata From This Page menüpont segítségével. A következő ábra már a metaadatok böngészését mutatja a Piggy Bank felhasználói felületén, ahol az XMP metaadatok kezeléséhez rendelkezésre áll a Piggy Bank teljes eszköztára.

4.5. Megvalósítás

A megvalósítás során a szerző a Piggy Bank kiterjesztéshez adott hozzá XMP támogatást. Az implementáció valójában két élesen elkülöníthető részből állt:

- a böngészőtől független XMP kinyerés,
- a fenti funkciót a böngészőbe integráló felhasználói felület.



4.3. ábra. A kinyert XMP metaadatok böngészése a Piggy Bank felhasználó felületén

4.4. ábra. A XMP kinyerés beállítási lehetőségei

4.5.1. Felhasználói felület

A Piggy Bank felhasználói felülete XUL-ban [12] készült. A XUL (XML User Interface Language) a Mozilla projektben kifejlesztett platformfüggetlen felhasználói felület leíró nyelv, amely elsősorban a Mozilla alkalmazásokat célozza meg. A Firefox böngésző és a hozzá rendelkezésre álló kiterjesztések teljes felhasználói felülete XUL-ban készült. A XUL használatához olyan további technológiák és szabványok alkalmazása szükséges, mint például a CSS, JavaScript és RDF. Egyik legfontosabb jellemzője, hogy olyan módon teszi lehetővé elemek kívülről történő hozzáadását egy felhasználói felülethez, hogy ehhez nem szükséges a kibővítendő felület definícióját módosítani.

A böngésző felületéhez az alábbi elemeket kellett hozzáadni:

- A böngésző Tools/Eszközök menüjébe beépülő XMP almenüt, amely az Extract XMP Metadata From This Page és az Options... menüpontokat tartalmazza (lásd a 4.2. ábrán).
- A beállítások megadására szolgáló XMP Options dialógusablakot (lásd a 4.4. ábrán).
- Egy képen vagy hiperhivatkozáson az egér jobb gombjával előhívható menühöz egy Extract XMP Metadata From Image illetve Extract XMP Metadata From Link menüpontot.

Az XMP kinyerést ténylegesen egy REST-stílusú webszolgáltatással [16] végezte el a program, amelynek a böngészőbe integrálása JavaScript nyelven történt.

4.5.2. XMP kinyerő webszolgáltatás

Az XMP kinyerő webszolgáltatást a HTTP GET metódus [14] segítségével lehetett igénybe venni: paraméterként egy URI-t kapott, eredményként pe-

dig az erőforrásból kinyert XMP metaadatokat adta vissza RDF/XML-ben. A fejlesztő szempontjából a megközelítés egy nyilvánvaló előnye, hogy az XMP kinyerés megvalósítása és tesztelése a böngésző kiterjesztéstől teljesen függetlenül történhetett. A felhasználó számára kedvező, hogy a kliens oldal pehelysúlyú, nem igényelte XMP kinyerő szoftverkomponensek telepítését.

A webszolgáltatás működése az alábbiakban foglalható össze:

1. Paraméterként egy URI-t kap, amely a feldolgozandó erőforrást azonosítja.

2. A HTTP HEAD metódus [14] segítségével meghatározza az erőforrás reprezentációjának MIME-tartalomtípusát [4] és méretét.

   - Ha az erőforrás nem található vagy nem támogatott a reprezentáció formátuma, akkor a hibát jelző megfelelő választ ad.
   - Hibát eredményez az is, ha a reprezentáció mérete nagyobb egy beállítható értéknél.

3. Megkezdődik a reprezentáció letöltése, amely során kinyerésre kerülnek a beágyazott XMP csomagok.

   - Ha vannak XMP csomagok, akkor ezek utófeldolgozása után egy a metaadatokat tartalmazó RDF/XML dokumentumot ad válaszként.
   - Ha nincsenek XMP csomagok, akkor az eredmény egy hibát jelző megfelelő válasz.

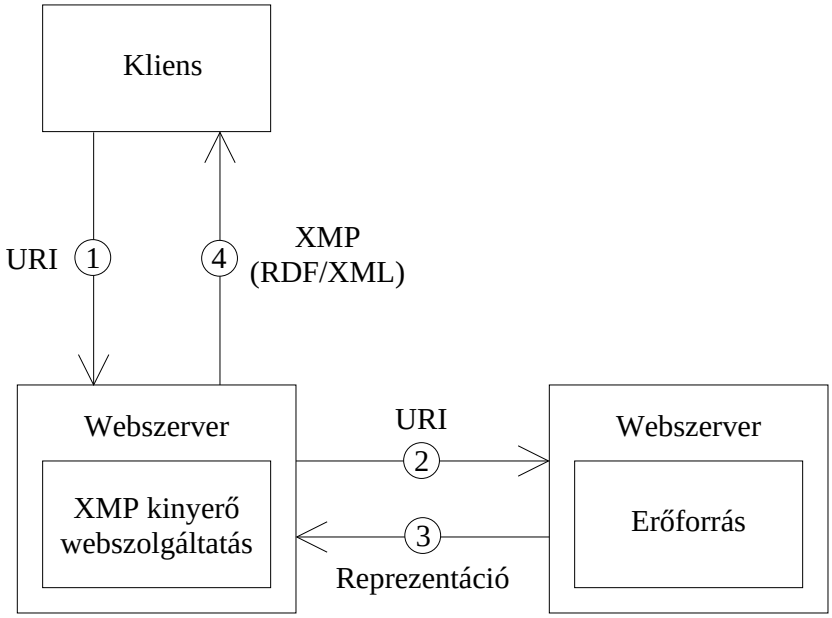
Az imént részletezett működést mutatja sematikusan a 4.5. ábra.

A webszolgáltatás Java-ban került megvalósításra a JAX-WS API-ra [2, 3] támaszkodva, működtetése pedig a szabad és nyílt forrású Apache Tomcat [6] segítségével történt. Maga a webszolgáltatás egyébként csupán egy felületet biztosított a következő szakaszban bemutatásra kerülő XMP kinyerő keretrendszerhez.

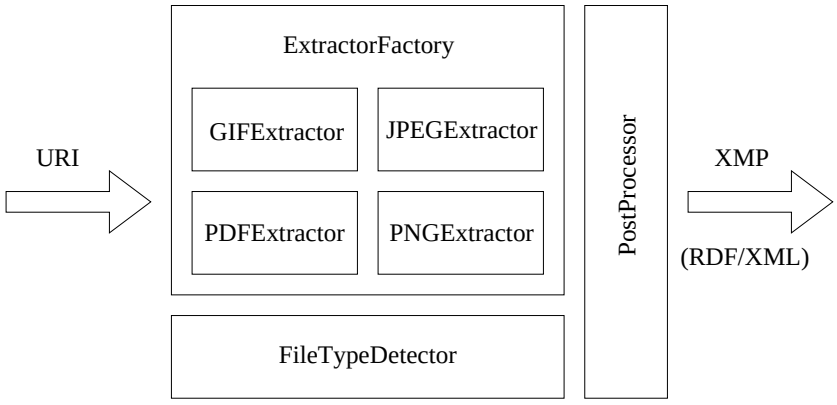
4.5.3. XMP kinyerő keretrendszer

A munka keretében a szerző kidolgozott egy XMP csomagok kinyerésére szolgáló saját Java osztálykönyvtárat, amelynek felépítése nagyon hasonló a 3.6. szakaszban tárgyalt RDF kinyerő keretrendszerhez, valójában annak őseként tekinthető. Szerkezetét a 4.6. ábra mutatja.

A keretrendszer lelkét az `ExtractorFactory` osztály és az `Extractor` interfész alkotja. Az utóbbit implementáló osztályok végzik az XMP csomagok



4.5. ábra. Az XMP kinyerő webszolgáltatás működése



4.6. ábra. Az XMP kinyerő keretrendszer

kinyerését, mindegyikük egy adott formátumot kezel. Az `ExtractorFactory` osztály adminisztrálja a rendelkezésre álló kinyerőket, egy metódushívással kérhető egy paraméterként adott URI-hoz az azonosított erőforrást feldolgozni képes kinyerő objektum. [11] alapján a GIF, JPEG, PNG és PDF formátumokhoz készültek saját XMP kinyerő osztályok.

Az `ExtractorFactory` osztály a MIME-tartalomtípus megállapításával választja ki a megfelelő kinyerőt egy URI-hoz, ehhez az alábbi működésű `FileTypeDetector` osztály ad támogatást. Az `ftp` és `http` URI sémák esetén a MIME-tartalomtípus megállapítása a webszolgáltatásnál leírt módon, a HTTP HEAD metódussal történik. Lokális állományokat azonosító URI-k, azaz a `file` URI séma használata esetén pedig a MIME-tartalomtípus meghatározásához a Unix-szerű operációs rendszerekben alapértelmezésben rendelkezésre álló, de más környezetekbe is adaptált `file` [1] parancs hívható segítségül.

Technikai okokból az XMP csomagokon egy utófeldolgozási lépést is végre kell hajtani. A kinyerés során kapható csomagok speciálisan határolt XML dokumentum-töredékek, amelyeket az RDF feldolgozók számára alkalmas formába kell alakítani. Például el kell távolítani a 4.1. ábrán látható csomag első és utolsó két sorát, amelyek hibát okozhatnak egy RDF feldolgozóval történő beolvasás során. Másrészt az ábra harmadik sorában látható `rdf:RDF` elemen meg kell adni az `xml:base` [15] attribútumot, amelynek értékeként a beágyazó erőforrás URI-ja kell hogy megjelenjen. Ez azért elengedhetetlen, mert az `rdf:Description` elemek `rdf:about` attribútumának értéke a csomagokban általában az üres karakterlánc.[1] A bázis-URI alapján minden relatív hivatkozás – esetünkben üres karakterlánc – „abszolút” URI-vá oldható fel.

A szerző adós még annak magyarázatával, hogy mi indokolta a szakaszban tárgyalt saját XMP kinyerő eszközök használatát az XMP Toolkit SDK helyett. Ennek egyik okát az SDK akkori felhasználási feltételei jelentették: a korábbi nyílt forrású verziók terjesztése egy olyan saját licenc hatálya alatt történt, amely nem volt kompatibilis más elterjedten használt szabad és nyílt forrású licencekkel. Ez a tény vitákat is generált a fejlesztői közösségben az SDK nyílt forrású projektekhez felhasználhatóságával kapcsolatban. A probléma időközben megszűnt, mivel az SDK utóbbi verzióit már a széles körben használt módosított BSD licence hatálya alatt adják közre. A szerző saját fejlesztésű megoldása egyébként a GNU GPL licenc hatálya alatt állt rendelkezésre. A másik probléma az volt, hogy az SDK csupán részleges Java támogatást nyújtott, nem tette lehetővé állományokból XMP csomagok kinyerését. (A formátum-specifikus kinyerőket tartalmazó XMPFiles kompo-

[1] Ilyen módon kerül kifejezésre, hogy az RDF hármasok a beágyazó erőforrást írják le.

nense csak Windows platformra állt rendelkezésre C++ implementációban.)

A bemutatott keretrendszer funkcionalitását tekintve nem összemérhető az XMP Toolkit SDK-val, hiszen csak XMP csomagok kinyerésére volt képes, nem támogatta csomagok manipulálását és állományokba beágyazását sem. Azonban egy kisméretű és hatékony alternatívát nyújtott olyan szabad és nyílt forrású Java projektekhez, amelyekben csupán a csomagok kinyerésére volt szükség.

4.6. Általánosítás

Vegyük észre, hogy a szerver oldalon az XMP csomagok kinyerését megvalósító programok a kliensek számára transzparens módon cserélhetők le tetszőleges egyéb RDF kinyerő eszközökre. Valójában tehát egy olyan kliens oldali funkcióról van szó, amelynek segítségével alkalmas webszolgáltatások rendelkezésre állása esetén a weboldalról elérhető tetszőleges erőforrások tetszőleges RDF reprezentációja böngészhető a Piggy Bank felületén.

Logikus lépés lett volna a Piggy Bank egy ilyen irányba továbbfejlesztése. Sajnos nem ez történt, a fejlesztés félbeszakadt, a legutóbbi verzió már a Firefox 3.x verzióival sem működőképes.

Irodalomjegyzék

[1] Fine Free File Command. URL http://darwinsys.com/file/.

[2] JSR 224: Java API for XML-Based Web Services (JAX-WS). URL http://jcp.org/en/jsr/detail?id=224.

[3] JAX-WS Reference Implementation. URL http://jax-ws.dev.java.net/.

[4] MIME Media Types. URL http://www.iana.org/assignments/media-types/.

[5] Piggy Bank. URL http://simile.mit.edu/piggy-bank/.

[6] Apache Tomcat. URL http://tomcat.apache.org/.

[7] Adobe XMP: Adding intelligence to media. URL http://www.adobe.com/products/xmp/.

[8] Adobe XMP Developer Center. URL http://directormx2.com/devnet/xmp/.

[9] XMP Specification Part 1: Data Model, Serialization, and Core Properties, 2010. URL http://directormx2.com/devnet/xmp/pdfs/XMPSpecificationPart1.pdf.

[10] XMP Specification Part 2: Additional Properties, 2010. URL http://directormx2.com/devnet/xmp/pdfs/XMPSpecificationPart2.pdf.

[11] XMP Specification Part 3: Storage in Files, 2010. URL http://directormx2.com/devnet/xmp/pdfs/XMPSpecificationPart3.pdf.

[12] XUL (XML User Interface Language). URL http://www.mozilla.org/projects/xul/.

[13] Dave Beckett. RDF/XML Syntax Specification (Revised). W3C Recommendation, 2004. URL `http://www.w3.org/TR/rdf-syntax-grammar/`.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Standard), 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`.

[15] Jonathan Marsh and Richard Tobin. XML Base (Second Edition). W3C Recommendation, 2009. URL `http://www.w3.org/TR/xmlbase/`.

[16] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2007. ISBN 978-0-596-80168-7.

III. rész

Csomagkezelés

5. fejezet

Csomagkezelés

5.1. Bevezetés

A számítógépes programok jellemzően számos különböző fajta állományból állnak, amelyek lehetnek például futtatható állományok, adatállományok vagy dokumentációt tartalmazó állományok, és amelyeket a telepítés során megfelelő helyekre kell másolni az állományrendszerben. Az alkalmazás futtatása előtt azonban további tevékenységek végrehajtása lehet szükséges.

Sok esetben a program forráskódban áll rendelkezésre, amelyet megfelelően kell lefordítani. Ez a feladat akár még egy szakember számára is kihívást jelenthet[1], ráadásul egy nagyobb alkalmazásnál sokáig tarthat. Egy többfelhasználós rendszerben az alkalmazás biztonságos futtatásához gyakran szükséges egy külön felhasználói fiók létrehozása. Biztonsági szempontból roppant kritikus az állomány jogosultságok és tulajdonjogok megfelelő beállítása is.

Hogy a telepítés bárki számára egyszerű feladat legyen, a szoftvergyártók telepítőprogramokat biztosíthatnak alkalmazásaikhoz, amelyek minimális felhasználói beavatkozást igényelve automatikusan végeznek el minden szükséges tevékenységet. Windows környezetben ez a megszokott gyakorlat.

A számítógépre telepített programok naprakészen tartása is megoldandó feladat. A Windows rendszerek a gyártó saját programjai és meghajtóprogramok esetében támogatják az automatikus frissítést. Egyéb alkalmazói programok is biztosíthatnak ilyen funkciót, azonban ezt a gyártó kell hogy

[1]Nagy segítséget jelentenek a fordítás-automatizáló eszközök (*build tools*). Számos programozási nyelvhez és környezethez állnak ilyenek rendelkezésre. Például Unix-szerű rendszerekben elterjedten használt ilyen program a `make`, az Apache Ant [3] és az Apache Maven [19] pedig elsősorban Java projektekhez kifejlesztett fordítás-automatizáló eszközök.

megfelelően implementálja, az automatikus frissítés nem a rendszer alapszolgáltatása.

Számos Unix-szerű operációs rendszer alapul szabad és nyílt forrású szoftvereken. Ilyenek a Linux-disztribúciók, amelyek közül a legelterjedtebbek közé tartozik például a Fedora [9] és az Ubuntu [17].[2] Egy **disztribúció** nem más, mint a Linux kernel és alkalmas formában összecsomagolt alkalmazások (**csomagok**) egy összessége. Ráadásul a legtöbb disztribúció ugyanazokat a szabad és nyílt forrású alkalmazói programokat tartalmazza, egymástól például a „csomagolás" módjában térhetnek el.

A szabad és nyílt forrású szoftverek használatának egy óriási előnye, hogy ezeket egységes módon lehet összecsomagolni, a csomagokat pedig hálózaton keresztül hozzáférhető központi helyeken, úgynevezett **tárolók**ban lehet elhelyezni. Ha minden alkalmazás egy helyen és egységes formában érhető el, egyszerűen megoldható a telepítés és frissítés. A modern Linux rendszerekben olyan eszközök adottak, amelyeknél néhány kattintás elegendő bármely a tárolóban lévő csomag telepítéséhez, az automatikus frissítés pedig alapszolgáltatás.

Ian Murdock, a Debian Linux-disztribúció alapítója fogalmazta meg azt a nevezetes kijelentést, hogy a fenti csomagkezelés a Linux-nak köszönhető legnagyobb előrelépés.[3]

A modern Linux-disztribúciókat és Unix-szerű operációs rendszereket sok esetben több ezer csomag alkotja. Ezekben a csomagok kezelését úgynevezett **csomagkezelő rendszerek** valósítják meg. Egy csomagkezelő rendszer többnyire egy adott csomagformátumot kezel, valamint számos olyan kötődő szolgáltatást nyújt, mint például a telepített csomagok automatikus frissítése vagy a telepítés során a csomagok közötti függőségek automatikus kezelése.

A csomagkezelés lehetővé teszi a teljes operációs rendszer naprakészen tartását, azonban nem csupán az operációs rendszerek kizárólagos funkciója, akár alkalmazások is használhatják és élvezhetik előnyeit. Az alkalmazás szinten megvalósított csomagkezelés lehetővé teszi a programok funkcióinak bővítését. Például a Firefox [11] böngészőhöz több száz olyan mindössze pár kattintással telepíthető **kiterjesztés** áll rendelkezésre, amelyek további funkciókat adnak hozzá a programhoz. Említhető a szabad és nyílt forrású R statisztikai és grafikai környezet [27] is, amely saját csomagformátumot és csomagkezelő rendszert használ.

A csomagkezelés előnyeit az alkalmazásfejlesztésben is élvezhetjük. Sok programozási nyelvhez és környezethez adottak olyan eszközök, amelyek cso-

[2]Ez nem zárja ki azt, hogy a disztribúciók nem szabad szoftvereket is tartalmazzanak.

[3]Murdock a [20] blogbejegyzésben teszi fel és válaszolja meg ennek megfelelően a „What's the single biggest advancement Linux has brought to the industry?" kérdést.

magkezelési funkciókat biztosítanak programkönyvtárak telepítéséhez. Ilyen például a szabad és nyílt forrású Apache Maven [19] projektkezelő eszköz, amely a Linux-rendszerekben megszokott csomagkezelést biztosítja Java projektekhez, forradalmasítva ezáltal a fejlesztés folyamatát.

Jól látható, hogy a csomagkezelés széles körben használt megoldás, amelynek számos eltérő felhasználása és megvalósítása létezik. Ugyanakkor közös vonásai is vannak a különböző megoldásoknak. Például a csomagok egy tipikus jellemzője, hogy sok-sok metaadatot tartalmaznak.

5.2. A csomagkezelés alapfogalmai

5.2.1. Szoftvercsomag

A **szoftvercsomag**, röviden **csomag** kifejezés egységnyi terjeszthető és telepíthető szoftvert jelent. Egy csomag legegyszerűbb esetben egyetlen archív állomány formájában adott. Minden csomagnak van neve és verziószáma, amelyek általában megjelennek az állománynévben is.

Egy csomag általában egy adott alkalmazást vagy szolgáltatást reprezentál, de nem minden esetben. Például a `filesystem` Fedora RPM csomag az operációs rendszer állományrendszerének könyvtárszerkezetét tartalmazza, amely nyilvánvalóan nem tekinthető alkalmazásnak.

Egy csomag tartalmazhat végrehajtható állományokat vagy forráskódot, valamint adatállományokat. **Bináris csomagok**nak nevezzük a végrehajtható állományokat és adatállományokat, **forráscsomagok**nak pedig a forráskódot és adatállományokat szolgáltató csomagokat. A bináris csomagok lehetnek csak egy adott számítógép-architektúrán működőképesek vagy platformfüggetlenek. Az előbbiek tipikusan bináris futtatható programokat tartalmaznak, az utóbbiak pedig szkripteket vagy számítógép-architektúra független bájtkódot.

Általában tartalmaznak metaadatokat is, mint például a csomag leírása, licence, a karbantartók elérhetősége és az előfeltételek. A csomagoknak hardver és szoftver követelményei lehetnek, a telepítéshez szükséges lehet más csomagok előzetes telepítése. Az utóbbi követelményeket **függőségek**nek nevezzük. A metaadatok beágyazásának módját a csomag állományformátuma határozza meg.

Sok csomagkezelő rendszer támogatja forrás- és bináris csomagok kezelését is, amelyek közül gyakran előnyben részesítik valamelyiket. A forrás csomagok szolgálhatnak kizárólag arra a célra, hogy adott célrendszerre készüljön belőlük bináris csomag.

5.2.2. Csomagkezelő rendszer

Csomagkezelő rendszernek egy olyan alkalmazást nevezünk, amely egységes módon és automatikusan teszi lehetővé csomagok telepítését. További kapcsolódó funkciókat biztosíthat, például támogathatja a rendelkezésre álló telepíthető csomagok keresését, a telepített csomagok automatikus frissítését. Általában egy adott csomagformátumot használ, mint például az RPM [32] vagy a `.deb` [36], és valamiféle adatbázisban tárolja a telepített csomagok metaadatait, amelynek lekérdezéséhez rendszerint lehetőséget biztosít.

Kényelmi szempontból nagy különbség lehet az egyes csomagkezelő rendszerek használatában. Emiatt gyakran megkülönböztetünk alacsony és magas szintű csomagkezelő eszközöket. Nem húzható éles választóvonal, de a magas szintű eszközök többnyire alacsony szintű eszközökre épülnek, például barátságosabb felhasználói felületet és további funkciókat biztosítva. Tipikusan a magas szintű eszközök által nyújtott funkció a függőségek automatikus telepítése.

5.2.3. Tároló

A csomagok terjesztéséhez gyakran tárolókat használnak. A Linux rendszerekben ez a megszokott módja a csomagok közzétételének. A csomagtároló, röviden **tároló** (**repository**) egy olyan hely, ahol telepíthető csomagok egy összessége áll rendelkezésre. A tárolókhoz a csomagkezelő rendszerek tipikusan hálózaton keresztül férhetnek hozzá, de akár adathordozókon (például CD vagy DVD lemezeken) is rendelkezésre állhatnak. Hálózati elérés esetén a tároló legegyszerűbb esetben lehet egy FTP vagy webszerver, de kifinomultabb hozzáférési módszereket is használnak.

A tárolók gyakran csupán a csomagok legújabb verzióját tartalmazzák. Jellemzően alkalmas **adatbázis**ban tárolják a rendelkezésre álló csomagok metaadatait, amelynek megvalósításához sok esetben közönséges szöveges állományokat használnak. A metaadatok a csomagokból kerülnek kinyerésre, amelyekhez képest további információkat nem szokás biztosítani. Extra szolgáltatásként a tároló támogathatja például a csomagok kategóriákba sorolását, amely a kiválasztást segíti. Az adatbázis hatékonyan használható fel kereséshez, egyik legfontosabb felhasználása a függőségek kinyerése.

5.2.4. Csomagok kapcsolatai

A csomagok kapcsolatokat deklarálhatnak más csomagokkal, amelyek metaadatként kezeltek a rendszerben. A csomagkezelő rendszerek egyik előnye ezeknek a **függőségek**nek nevezett kapcsolatoknak a kezelése.

Általában többféle kapcsolat is kialakítható, a legtipikusabb azt fejezi ki, hogy a deklaráló csomag használatához rendelkezésre kell hogy álljon a megadott csomag, amelyet a telepítés során a csomagkezelő rendszer ellenőriz. Általában a magas szintű csomagkezelő eszközök képesek a megkövetelt hiányzó csomagok telepítését automatikusan elvégezni. Egy csomag nem csupán kötelezőként írhatja elő egy másik rendelkezésre állását, megengedett lehet ajánlás kifejezése is.

A függőség fogalma használható csomagok közötti tetszőleges kapcsolat szinonimájaként, de bizonyos csomagkezelő rendszerekben csak ez előbbi fajta kapcsolatokat nevezik függőségeknek.

Sok csomagkezelő rendszer támogatja csomagok ütközését kifejező kapcsolat megadását, amely annak jelzésére szolgál, hogy a benne szereplő csomagokat nem lehet egy rendszerbe telepíteni.

A kapcsolatokban gyakran meg lehet adni verziószámokat és relációs operátorokat is, amelyekkel korlátozás írható elő a csomag verziószámára. A csomagkezelő rendszerek némelyike lehetővé teszi olyan kapcsolatok használatát is, amelyeket csak adott feltételek teljesülése esetén vesz figyelembe.

A forrás- és bináris csomagokat is kezelő rendszerek külön függőségeket biztosíthatnak a két fajta csomaghoz.

5.2.5. Verziószámok

Szoftvertermékek kiadásainak megkülönböztetésére szolgálnak a **verziószámok**, amelyek a különböző kiadásokhoz rendelt egyedi azonosítók.[4] Gyakran valóban számok alakját öltik, de általánosan csak annyi mondható, hogy olyan szimbólumsorozatokkal ábrázolhatók, amelyek többnyire számjegyeket is tartalmaznak, szerkezetük és megjelenésük azonban változatos lehet. Az azonosításon túl tájékoztathatnak a szoftver fejlesztésének állapotáról, kifejezhetik az egyes kiadások közötti eltéréseket.

A gyártók akár teljesen önkényesen választhatnak verziószámot minden egyes kiadáshoz, de általában valamilyen rendszert alkalmaznak, amely meghatározza a formát és használat módját. A számozási rendszerek általában lehetővé teszik rendezés értelmezését a verziószámok halmazán.

Gyakoriak a szimbólumsorozatokból álló strukturált és a legtöbb esetben $v_1.v_2.\cdots.v_n$ formában ábrázolt verziószámok, ahol minden v_i az üres szótól különböző megfelelő szimbólumsorozat. Tipikusan decimális számjegyek és betűk megengedettek a sorozatokban. Nagyon eltérő lehet azonban a sorozatok száma, értelmezése és kezelése.

[4]Természetesen nem csak szoftvereknek lehet verziószáma, hanem például dokumentumoknak is. Fontos szerepe van továbbá a fogalomnak a verziókezelő rendszerekben, amelyekben állományok állapotainak azonosításához használtak.

A fenti verziószámok egy lehetséges értelmezésében a szimbólumsorozatok a kiadások közötti eltéréseket reprezentálják. Széles körben elterjedtek a főszámból és alszámból álló, legtöbbször $v_{\text{major}}.v_{\text{minor}}$ alakban megjelenített verziószámok, ahol a v_{major} sorozat a főszám, a v_{minor} sorozat pedig az alszám. Adott szoftverhez tartozó verziószámokban különböző főszámok a megfelelő kiadások közötti lényeges eltéréseket jelentenek. Azonos főszámok esetén az alszámok eltérései csupán kisebb változásokra utalnak. A főszám és alszám mintájára a verziószám további sorozatokat is tartalmazhat, amelyek a számozási rendszerben alkalmas nevet kapnak.[5] Ábrázolásban mindig a főszám kerül a bal szélre, amelyet az alszám követ, végül az esetleges további sorozatok jönnek a reprezentált eltérések mértéke szerint csökkenő sorrendben.

Az induló verziószám tetszőlegesen választható. Új kiadáshoz azonosító a legutóbbi kiadás verziószáma alapján rendelhető, benne a megfelelő sorozat „növelésével" és a sorrendben ezt következő sorozatok alkalmas kezdőértékről újraindításával. A sorozatok tekinthetők egész számoknak, ekkor a növelés értelemszerűen végezhető el, például az 1.0.9 verziót követheti a 2.0.0, 1.1.0 vagy 1.0.10 verzió. Eltérő megoldások is léteznek, nem szokványosan történik a növelés például a TEX esetében, ahol a verziószámok a 3.1 kiadás óta π-hez tartanak: csak az alszám változik, amely mindig a π soron következő számjegyével bővül a végén [33].

Közvetlenül egymást követő kiadások verziószámai között között nagy „ugrások" lehetnek. Például a Linux kernel 0.12 verzióját a 0.95 számú váltotta fel.[6] Itt említhető a Netscape böngésző, amelynek $4.x$ verziószámú kiadásait közvetlenül a $6.x$ számú verziók követték.[7]

Verziószámok képzése történhet dátumokból, az ábrázolásnál általában szabványos dátumformátumokat használnak. Dátumok komponensei megjelenhetnek szimbólumsorozatokból álló strukturált verziószámokban is. Ilyen számokat viselnek például az Ubuntu Linux-disztribúció kiadásai, amelyek közül a legutóbbi a 2010 októberében megjelent 10.10 verziószámú.[8]

A verziószám része lehet a fejlesztés állapotára utaló olyan betűsorozat, mint például `alpha`, `beta` vagy a *release candidate* kifejezést rövidítő `rc`.

A csomagkezelő rendszerek minden csomaghoz megkövetelik verziószám rendelkezésre állását, amelyet többnyire közönséges karakterláncokként kapnak és tárolnak. A csomagkezelő rendszer határozza meg a használható

[5] Néhány elterjedten használt elnevezés: *build number*, *release number*, *patch level*.

[6] Mindkét verzió 1992. évi kiadású és a korai kernel verziók társaságában a `http://www.kernel.org/pub/linux/kernel/Historic/old-versions/` címen érhető el.

[7] A Netscape böngésző kiadásainak listáját lásd a `http://browser.netscape.com/releases` címen.

[8] Az Ubuntu kiadásait lásd a `https://wiki.ubuntu.com/Releases` címen.

verziószámok formátumát. A csomag verziószámának alapjául a tartalmazott szoftver verziószáma szolgál. Az eredeti verziószám módosítása szükséges akkor, ha az nem megfelelő a rendszer számára.

A csomagkezelő rendszerek működésének fontos eleme a verziószámok összehasonlítása. A rendszer összehasonlításban azonosnak tekinthet eltérően ábrázolt verziószámokat, tehát a verziószámok halmazán egy ekvivalencia reláció definiálható. A rendszer egy rendezést is értelmez az verziószámok halmazán. Ennek egy tipikus megvalósítása az összehasonlítandó verziószámok komponensekre bontásán és a komponensek balról jobbra haladva történő páronkénti összehasonlításán alapul.

Nem könnyű megfelelő rendezés definiálása. Mivel a verziószámokat karakterláncok reprezentálják, kézenfekvő gondolat a lexikografikus rendezés választása. Ez sajnos már a lehető legegyszerűbb, csak decimális számjegy és pont karaktereket tartalmazó verziószámok esetén sem megfelelő, hiszen például a lexikografikus rendezés szerint az 1.10 verzió kisebb, mint az 1.2 számú.

Verziószámok összehasonlításával dönthető el, hogy egy telepített csomagot szükséges-e egy másik verzióra cserélni. Csomagfrissítés során általában csak akkor kívánatos a csere, ha a telepített csomag verziószámánál nagyobb a másiké. Az előbbi esetben a csomag lecserélésére az **upgrade** kifejezést használják. Egy csomagot akár egy kisebb verziószámúra is lehet cserélni, ennek neve **downgrade**, azonban ezt nem minden rendszer támogatja.

Fontos hangsúlyozni, hogy sokféle verziószámozási rendszer létezik, amelyek sajátos módon értelmezhetik a rendezést. A csomagkezelő rendszerek azonban mindig egy adott rendezést használnak, amely az összehasonlításnál az eredeti számozási rendszernek nem megfelelő eredményt adhat. Megoldásként a problémás verziószámokat a csomagkezelő rendszer számára megfelelő alakba lehet átformázni. Egy másik lehetőség a verziószám részeként epoch[9] használata.

5.3. Csomagkezelés operációs rendszerekben

A fejezet az operációs rendszerekben használt csomagkezelési megoldásokról ad áttekintést a teljesség igénye nélkül. Valamennyi rendszerben többékevésbé ugyanazokat a szolgáltatásokat biztosítják csomagkezelés néven. El-

[9]Több csomagkezelő rendszer támogatja a verziószám első komponenseként úgynevezett epoch megadását, amely egy nemnegatív egész szám, és amelyet tipikusan egy : karakter választ el a verziószám további részétől. Kifejezetten arra szolgál, hogy lehetővé tegye olyan kivételes verziószámok kezelését, amelyek esetén egyébként a csomagkezelő rendszer által használt rendezés nem megfelelő eredményt adna.

térések a megvalósításban vannak, így például különbözhet a csomagok állományformátuma és a metaadatok lokális tárolásának módja.

5.3.1. Unix-szerű operációs rendszerek

A csomagkezelés tipikusan alapszolgáltatás a szabad és nyílt forrású Unix-szerű operációs rendszerekben, amelyek pontos számát nehéz lenne megmondani, de bizonyosan több száz ide sorolható disztribúció létezik.[10] Az általuk használt különböző csomagkezelési megoldások száma azonban lényeges kevesebb, emiatt gyakran éppen a csomagkezelés képezi a disztribúciók csoportosításának alapját.

5.3.1.1. Debian-alapú Linux-disztribúciók

Sok további Linux-disztribúció származik a Debian operációs rendszerből. A Debian és valamennyi belőle származtatott operációs rendszer csomagkezelésének lelke a `.deb` állományformátum és a `dpkg` csomagkezelő program. Az utóbbi egy olyan alacsony szintű parancssoros eszköz, amelyre további csomagkezelő programok épülnek. C programozási nyelven készült, a csomag metaadatok lokális tárolásához közönséges szöveges állományokat használ.

A `dpkg` előtétek közül történetileg az első a ma már valószínűleg csak kevesek által használt `dselect` program, amely egy szöveges módú interaktív felhasználói felületet biztosít. Többféle forrásból teszi lehetővé csomagok telepítését, automatikusan kezeli a függőségeket, képes a telepített csomagok frissítésére, azonban kezelése meglehetősen bonyolult.

A `dselect`-nél kényelmesebb az Advanced Package Tool (APT) parancssoros eszköz használata, amely maga is további csomagkezelő alkalmazások alapjául szolgál: az `aptitude` szöveges, a `synaptic` [28] pedig grafikus felhasználói felületet ad hozzá.

A Debian rendszer jól bevált csomagkezelő eszköztárát tőle idegen környezetekbe is átvették. Például a Fink projekt [10] keretében a Darwin és Mac OS X operációs rendszerekre adaptálnak szabad és nyílt forrású Unix szoftvereket. Biztosítanak csomagkezelést is, amelyhez a `dpkg`, `dselect` és APT programokat használják.

5.3.1.2. RPM-alapú rendszerek

Az RPM betűszó egyidejűleg jelent egy csomagkezelő programot és állományformátumot. Egykor a Red Hat Package Manager kifejezés rövidítése volt,

[10] A Linux-disztribúciók valamint további szabad és nyílt forrású operációs rendszerek felsorolását lásd például a [8] weboldalon.

mai feloldása azonban RPM Package Manager. Az RPM eredetileg a Red Hat cég fejlesztése, azonban szabad és nyílt forrású. Sok Linux-disztribúció valósítja meg általa a csomagkezelést, amelyeket ezért összefoglaló néven RPM-alapú Linux-disztribúcióknak is hívnak. Ilyenek például a CentOS, Fedora, Mandriva Linux, openSUSE, Oracle Enterprise Linux és a Red Hat Enterprise Linux.

Az RPM Package Manager képességeiben a Debian rendszer `dpkg` programjához hasonló alacsony szintű parancssoros eszköz, amelynek alapja egy C programkönyvtár. A csomag metaadatok lokális tárolása Berkeley DB [5] adatbázisokban történik. Egy rá épülő magasabb szintű parancssoros eszköz a Yum [2], amelyhez létezik grafikus felhasználói felület is a Yum Extender [30] képében. A Yum és Yum Extender Python nyelven készültek.

Az RPM nem csupán Linux rendszerekben áll rendelkezésre, hanem megtalálta útját más Unix-szerű környezetekbe is, lásd például az OpenPKG kapcsán az 5.3.2.3. szakaszban leírtakat.

5.3.1.3. Gentoo-alapú Linux rendszerek

A Gentoo [14] egy szabad és nyílt forrású Linux-disztribúció, amelyből több operációs rendszer is származik. Csomagkezelő rendszerének neve Portage [37]. Fő jellemzője forráscsomagok elsődleges használata, melyekből a telepítés során a lokális gépre optimalizálva történik a bináris programok előállítása. Mivel a fordítás nagyméretű programok esetében időigényes lehet, adva van bináris csomagok használatának lehetősége is, de a projekt keretében nem biztosítanak adott platformokra előre elkészített bináris csomagokat.

A Gentoo csomagok úgynevezett `.ebuild` állományok formájában adottak, amelyek speciális shell szkriptek. Metaadatokat tartalmaznak környezeti változókban, valamint a telepítéshez szükséges parancsokat. A csomagok forrásai ténylegesen tömörített archív állományokban állnak rendelkezésre. A telepítési folyamat során során az `.ebuild` szkriptben adott helyről letöltésre és kibontásra kerül a forrásokat tartalmazó állomány, majd végrehajtásra kerülnek a fordításhoz és a telepítéshez szükséges további parancsok.

A Portage csomagkezelő rendszer Python és Bash nyelven implementált. Az `ebuild` program egy alacsony szintű interfészt valósít meg a rendszerhez, feladata az `.ebuild` állományok kezelése. Egy magasabb szintű parancssoros eszköz az `emerge`, amely kezeli csomagok függőségeit és képes a telepített csomagok frissítésére is. Vannak természetesen grafikus felhasználói felületek is a Portage rendszerhez, ilyenek például a Portato [25] és a Porthole [26] programok.

A Gentoo rendszerhez léteznek a Portage alternatíváját nyújtó csomag-

kezelő rendszerek is, mint például a Paludis [22] és a pkgcore [23], amelyek szintén az `.ebuild` állományokat használják. A [31] dokumentum tárgyalja részletesen az `.ebuild` állományokat és tárolókat, amelyet a Gentoo csomagkezelő rendszerek szabványként kell hogy adaptáljanak.

5.3.1.4. FreeBSD

A szabad és nyílt forrású FreeBSD [12] operációs rendszer kétféle módon is lehetővé teszi alkalmazások telepítését [34]:

- Forrás csomagokból, amelyek neve a rendszerben port.
- Bináris csomagokból, amelyeket egyszerűen csomagoknak neveznek a rendszerben.

Külön módszerek és eszközök szolgálnak a forrás- és bináris csomagok kezelésére.

Minden portot több különálló állomány alkot. A rendelkezésre álló portok összességét port gyűjteménynek (*ports collection*) nevezik. Ez egy olyan könyvtárszerkezet, amelyben minden portot egy külön alkönyvtár tartalmaz. A teljes port gyűjtemény rendelkezésre áll lokálisan a telepítéshez. Mivel a portok száma jelenleg húszezernél több, ez csak úgy lehetséges, hogy a forrásállományok ténylegesen nincsenek jelen minden port könyvtárában. A portokhoz lokálisan csak metaadatokat tartalmazó és a telepítést elvégezni képes állományok adottak, a telepítés során kerülnek letöltésre a források, ezután történik meg a fordítás és kerülnek végrehajtásra a telepítés egyéb szükséges lépései.

A legtöbb port elérhető csomag formájában is, amely egyetlen tömörített archív állomány, benne metaadatokat tartalmazó és a csomagot alkotó állományokkal.

Az eszközök portok és csomagok esetében is automatikusan kezelik a függőségeket.

5.3.2. Platformfüggetlen megoldások

5.3.2.1. PackageKit

A szabad és nyílt forrású PackageKit [1] egy meta-csomagkezelőnek tekinthető alkalmazás. Platformfüggetlen, azonban olyan a freedesktop.org [13] projekt keretében fejlesztett szabványokon alapul, mint például a D-Bus [7] és a PolicyKit [24], amelyeket tipikusan a Unix-szerű rendszerek támogatnak. Egységes felületet biztosít tetszőleges csomagkezelő rendszerek eléréséhez. A

PackageKit önmagában nem működőképes, olyan csomagkezelő rendszerekkel tud együttműködni, amelyekhez rendelkezésre áll megfelelő illesztőfelület. Számos csomagkezelő rendszerhez vannak ilyenek, például az APT, Portage és Yum is támogatottak. Mivel alapvetően egy felület, értelemszerűen ezek eszközrendszerét használja.

Alapértelmezésben egy parancssoros program áll rendelkezésre csomagok kezeléséhez, a gnome-packagekit a GNOME, a KPackageKit pedig a KDE környezethez fejlesztett grafikus felhasználói felület.

5.3.2.2. Image Packaging System

Az Image Packaging System (IPS), más néven pkg(5) [18] az OpenSolaris közösség és a Sun Microsystems közös fejlesztésű csomagkezelő rendszere. Az IPS csomagokat állományok, könyvtárak, eszközmeghajtók, szimbolikus linkek és metaadatok alkotják. Minden csomag bináris, fordítás nem része a rendszer működésének.

Az IPS különlegessége a csomagkezelő rendszerek többségéhez képest az, hogy jelenleg nem definiált a csomagokhoz állományformátum. Működéséhez elengedhetetlenek a tárolók, amelyek elérése hálózaton keresztül történik. A csomagok telepítésre kizárólag tárolókban állnak rendelkezésre. A rendszer lehetővé teszi saját tárolók üzemeltetését, csomagok létrehozását tárolókban, csomagok telepítését tárolókból és a telepített csomagok frissítését.

Az IPS a nyílt forrású OpenSolaris operációs rendszer csomagkezelő rendszere. Azért került mégis a platformfüggetlen megoldások közé, mert Update Center Toolkit [29] néven több különböző platformra adaptálva is rendelkezésre áll. Ebben a formában azonban a rendszer már nem alkalmas operációs rendszer szintű csomagkezelés megvalósítására, alkalmazásokhoz adható általa csomagkezelés. Az Update Center Toolkit például a GlassFish [15] alkalmazásszerverhez használt.

5.3.2.3. OpenPKG

Az OpenPKG [21] egy szabad és nyílt forrású platformfüggetlen RPM-alapú csomagkezelő rendszer Unix rendszerek számára, amelyben jelenleg nagyjából 1500 telepíthető csomag áll rendelkezésre. A platformok közötti hordozhatóság biztosításához a rendszer forrás RPM csomagokat használ. Telepítés során a letöltött forráscsomagokból fordítás révén készülnek bináris RPM csomagok a célrendszerre, amelyek ilyen módon csak átmenetileg léteznek.

5.3.3. Windows

5.3.3.1. Nem szabad és nyílt forrású megoldások

Google Pack [16] néven kínál a Google Windows platformra saját és partnercégektől származó olyan alkalmazásokat, mint például az Adobe Reader, Google Chrome, Google Earth, Mozilla Firefox és a Skype. A gyűjteményt alkotó szoftverek telepítéséhez egy Google Updater nevű csomagkezelő program áll rendelkezésre.

5.3.3.2. Szabad és nyílt forrású megoldások

Több szabad és nyílt forrású projekt tűzte ki célul, hogy Windows környezetben a Linux rendszerekben megszokott csomagkezelést biztosítsanak alkalmazások telepítéséhez.

Egy ilyen program az Appupdater [4], amely hálózaton keresztül elérhető tárolókból teszi lehetővé Windows alkalmazások telepítését és karbantartását, beleértve az automatikus frissítést. Az alapértelmezésben használt tárolóban jelenleg közel kilencven népszerű alkalmazás áll rendelkezésre, amelyek nem mindegyike szabad és nyílt forrású. Saját tároló üzemeltetését is támogatja. A tárolókban XML dokumentumokat használnak a kínált alkalmazások nyilvántartásához.

2010 tavaszán került bejelentésre a Microsoft támogatását élvező Common Opensource Application Publishing Platform (CoApp) [35, 6] közösségi projekt indulása, amelynek keretében csomagkezelő rendszert fejlesztenek nyílt forrású Windows alkalmazásokhoz. A fejlesztők 2011 elejére ígérik a rendszer első béta teszt kiadását.

Irodalomjegyzék

[1] PackageKit. URL http://www.packagekit.org/.

[2] Yum Package Manager. URL http://yum.baseurl.org/.

[3] Apache Ant. URL http://ant.apache.org/.

[4] Appupdater. URL http://www.nabber.org/projects/appupdater/.

[5] Oracle Berkeley DB. URL http://www.oracle.com/technology/products/berkeley-db/.

[6] Common Opensource Application Publishing Platform (CoApp). URL http://coapp.org/.

[7] D-Bus. URL http://freedesktop.org/wiki/Software/dbus.

[8] DistroWatch. URL http://distrowatch.com/.

[9] Fedora. URL http://fedoraproject.org/.

[10] Fink. URL http://www.finkproject.org/.

[11] Firefox web browser. URL http://www.mozilla.org/firefox/.

[12] The FreeBSD Project. URL http://www.freebsd.org/.

[13] freedesktop.org. URL http://www.freedesktop.org/.

[14] Gentoo Linux. URL http://www.gentoo.org/.

[15] GlassFish. URL https://glassfish.dev.java.net/.

[16] Google Pack. URL http://pack.google.com/.

[17] Ubuntu. URL http://www.ubuntu.com/.

[18] Image Packaging System. URL http://opensolaris.org/os/project/pkg/.

[19] Apache Maven. URL http://maven.apache.org/.

[20] How package management changed everything, 2007. URL http://ianmurdock.com/solaris/how-package-management-changed-everything/. Blog entry.

[21] OpenPKG. URL http://www.openpkg.org/.

[22] Paludis. URL http://paludis.pioto.org/.

[23] pkgcore. URL http://www.pkgcore.org/.

[24] PolicyKit. URL http://freedesktop.org/wiki/Software/PolicyKit.

[25] Portato. URL http://necoro.eu/portato/.

[26] The Porthole Portage Frontend. URL http://porthole.sourceforge.net/.

[27] The R Project for Statistical Computing. URL http://www.r-project.org/.

[28] Synaptic Package Manager. URL http://www.nongnu.org/synaptic/.

[29] Multi-platform Packaging for Layered Distros. URL http://wikis.sun.com/display/IpsBestPractices/.

[30] Yum Extender. URL http://www.yum-extender.org/.

[31] Stephen P. Bennett and Ciaran McCreesh. Package Manager Specification, 2010. URL http://distfiles.gentoo.org/distfiles/pms-3.pdf.

[32] Eric Foster-Johnson. RPM Guide, 2005. URL http://rpm5.org/docs/rpm-guide.pdf.

[33] Donald E. Knuth. The future of TEX and METAFONT. *TUGboat*, 4: 489, 1990.

[34] The FreeBSD Documentation Project. Installing Applications: Packages and Ports. In *FreeBSD Handbook*. 2010. URL http://www.freebsd.org/doc/en/books/handbook/.

[35] Garrett Serack. The Common Opensource Application Publishing Platform (CoApp), 2010. URL `http://blogs.msdn.com/b/garretts/archive/2010/03/31/the-common-opensource-application-publishing-platform-coapp.aspx`.

[36] The Debian Policy Mailing List. *Debian Policy Manual*, 2010. URL `http://www.debian.org/doc/debian-policy/`. version 3.8.4.0.

[37] Sven Vermeulen. A Portage introduction. In *Gentoo Handbook*. 2010. URL `http://www.gentoo.org/doc/en/handbook/`.

6. fejezet

Linked Data

6.1. „Adat web” építése

A Linked Data kifejezés és a mögötte rejlő elgondolás Tim Berners-Leetől származik [29]. A szókapcsolatnak a magyar nyelvben még nem létezik általánosan elfogadott megfelelője. Egy alkalmas átültetése nyelvünkre a szó szerinti fordításnak tekinthető „kapcsolt adatok” kifejezés, amely megfelelően tükrözi az elképzelés lényegét.

Az RDF egy egyszerű és általános keretrendszert biztosít erőforrások leírásához, amely az erőforrásokhoz rendelt URI-kat formális azonosítóként tekinti. A hiperszöveg webnek azonban alapszolgáltatása a dokumentumok közötti navigálhatóság, amelyhez elengedhetetlen az URI hivatkozás-feloldás. A Linked Data kifejezés az RDF gyakorlati használatának egy módját jelenti, amely lehetővé teszi egy olyan „adat web” létrehozását, amelyen éppen úgy lehet navigálni kapcsolatok követésével, mint a hiperszöveg weben.

Tim Berners-Lee néhány olyan szabályt fektet le az elképzelést vázoló [29] dokumentumban, amelyek betartásával „adat web” építhető.[1] A szabályok az URI-k használatára vonatkoznak: kizárólag a http URI séma megengedett, valamint minden URI-val az erőforrás leírása kell hogy elérhető legyen. Ha egy erőforrás leírását kell szolgáltatni, akkor olyan RDF kijelentéseket kell visszaadni, amelyekben az erőforrást azonosító URI alany vagy tárgy pozícióban szerepel, de lehet akár további kapcsolódó erőforrásokat leíró RDF hármasokat is.

A továbbiakban **a Linked Data kritériumoknak megfelelő URI**-nak egy olyan HTTP URI-t nevezünk, amelynek hivatkozás-feloldása során az

[1] A Tim Berners Lee-féle alapelveken túl érdemes figyelembe venni [35] az RDF használatára javasolt korlátozásait. Gyakorlati megfontolásokból nem ajánlott bizonyos RDF konstrukciók használata, ilyenek az üres csomópontok, a RDF konténerek és kollekciók, valamint a tárgyiasítás.

URI által azonosított erőforrás RDF leírása nyerhető reprezentációként.

RDF linkeknek hívunk olyan RDF kijelentéseket, amelyekben az alany és a tárgy URI, azaz amelyeket három URI alkot. Az RDF linkek két erőforrás közötti tipizált kapcsolatokat reprezentálnak.

A hipertext weben használt hivatkozások a hivatkozást tartalmazó forrás dokumentum és a célként megadott erőforrás között definiálnak kapcsolatot. Ezzel szemben az RDF linkek lehetővé teszik különböző adatforrások közötti kapcsolatok leírását a szóban forgó adatforrásoktól függetlenül, hasonlóan az XLink [42] szabvány kiterjesztett linkjeihez.

Egy **RDF link követése** az alany vagy tárgy pozícióban lévő URI hivatkozás-feloldását jelenti. Ha RDF publikáláshoz kizárólag a Linked Data kritériumoknak megfelelő URI-kat használunk, akkor az RDF linkek követése a hipertext weben megszokott navigálás élményét adja.

Azok az RDF linkek a legértékesebbek, amelyek különböző adatforrások között definiálnak kapcsolatokat. Miattuk lehet az „adat web" több izolált RDF gráfok összességénél. A globális „adat web" épüléséhez saját magunk új RDF linkek létrehozásával járulhatunk hozzá. Általában törekedni kell továbbá új URI-k bevezetése helyett létező erőforrásokat azonosító URI-k használatára. Olyan közismert és elterjedten használt szókészletekkel célszerű leírni az erőforrásokat, mint például a Dublin Core [7, 6] és a FOAF [39]. Megfelelő URI-k keresését segítik az olyan szemantikus web keresőmotorok is, mint például a Sindice [54] vagy a Swoogle [45].

Mivel a hagyományos böngészők tipikusan nem képesek RDF tartalom megfelelő megjelenítésére, **Linked Data böngészők** szolgálnak az „adat weben" navigálásra. Ilyen például a Tabulator [25, 31], amely két formában áll rendelkezésre: a Firefox böngészőbe beépülő kiterjesztésként, és egy olyan JavaScript könyvtárként, amellyel a böngésző funkció weboldalakba integrálható. Az OpenLink Data Explorer Extension [19] szintén egy olyan Firefox kiterjesztés, amely lehetővé teszi az „adat web" böngészését. Egy szerver oldali alternatíva a Marbles [16] webalkalmazás, amely egy webes felületet ad Linked Data böngészéshez. Mindhárom eszköz szabad és nyílt forrású.

6.2. Nem információ erőforrások azonosítása

Az RDF gyakorlati felhasználásaiban felmerülő probléma nem információ erőforrásokhoz megfelelő URI választása. Kézenfekvő megoldás olyan URI séma használata, amelynél az URI formális azonosítóként funkcionál. Nem értelmezett hivatkozás-feloldás például az `urn:isbn:9630774534` és `urn:ietf:std:3986` URI-k esetében. Mindkettő egyértelműen azonosít egy megfelelő dolgot, előbbi egy könyvet, utóbbi pedig egy IETF szabványt.

A nehézséget annak megválaszolása jelenti, hogy hogyan lehet hozzájutni az erőforrásokat leíró RDF kijelentésekhez akkor, ha csak a fenti két URI áll rendelkezésre.

Azért népszerűek mégis az ilyen URI-k, mert az allokáció – erőforráshoz URI rendelése – bármiféle felelősség nélkül végezhető el, így például nem kell törődni a hivatkozás-feloldhatóság biztosításával. HTTP URI-k alkalmazása azért rögös út sok felhasználó számára, mert a megfelelő hivatkozás-feloldás biztosításához szükség lehet rendszeradminisztrátori közreműködésre, például akkor, ha webszerver beállításokat kell módosítani.

HTTP URI-k RDF alkalmazásokban történő gyakorlati használatáról szól a [53] dokumentum, amely megoldásként az alább is tárgyalt hash URI-k és a 303-as átirányítás használatát mutatja be fogalmi szinten, egyben útmutatást ad adott esetben a megfelelőbb módszer kiválasztásához.

A [32] dokumentum ugyan kifejezetten az RDF Schema [38] és az OWL [21] kapcsán tárgyalja a témát, de úgy veszi számba a HTTP URI-k használatánál felmerülő megoldásokat, hogy az Apache webszerverhez [10] megadja ezek megfelelő beállításának módját is.

A téma szempontjából is iránymutató Tim Berners-Lee írása [28], amely általában foglalkozik az URI-k megfelelő kialakításával, megadva a gyakorlati szempontból jó URI-k ismérveit.

6.2.1. Hash URI-k

Nem információ erőforrások azonosítására használni lehet úgynevezett **erőforrásrész-azonosítót tartalmazó URI**-t is, amelyet gyakran **hash URI**-nak hívnak. Az **erőforrásrész-azonosító** az URI-k egy opcionális része. Ha egy URI tartalmaz `#` karaktert, akkor így nevezzük az ezt követő részét.[2] Például a `http://example.com/foaf.rdf#me` URI erőforrásrész-azonosítója a `me` karakterlánc.

Az erőforrásrész-azonosító egy másodlagos erőforrás kijelölésére szolgál. Értelmezését mindig a hivatkozás-feloldás során kapható reprezentáció MIME-tartalomtípusa [17] határozza meg, jelentheti például az erőforrás egy részét. Az értelmezés a kliens oldalon elvégzendő feladat, ezért az erőforrásrész-azonosító hivatkozás-feloldás során mindig eltávolításra kerül. Értelmezés a feloldás során nyert reprezentáció átvitele után lehetséges.

Az RDF alapfogalmait definiáló [50] szabvány rendelkezik az erőforrásrész-azonosítók kezeléséről `application/rdf+xml` média típusú reprezentációk esetében, amelyek RDF/XML szintaxisban ábrázolnak RDF gráfokat. Eszerint az erőforrásrész-azonosítót tartalmazó URI-k ugyanazt jelentik, mint

[2] Minden URI-ban legfeljebb egy literális `#` karakter megengedett.

előfordulásaik a hivatkozás-feloldás eredményeként kapható reprezentációban. Ez a jelentés tetszőleges lehet, amelyet a nyert reprezentáció RDF kijelentései határoznak meg. Semmiféle kapcsolat nem feltételezhető azonban olyan erőforrásrész-azonosítót tartalmazó URI-k között, amelyek csak az erőforrásrész-azonosítóban különböznek.

A hash URI-k elterjedten használtak ilyen módon RDF szókészletekben és OWL webontológiákban osztályok és tulajdonságok azonosítóiként, ahol az erőforrásrész-azonosító egy olyan URI-t követ, amellyel a definíciót tartalmazó RDF/XML reprezentáció nyerhető. Általában kézenfekvő megoldást jelentenek „kapcsolt adatok" statikus RDF/XML állományokban publikálásához.

6.2.2. 303-as átirányítás

A címben szereplő szám a HTTP/1.1 [44] egy állapotkódját jelenti. Nem információ erőforrást azonosító URI esetén a webszerver átirányítást végezhet egy olyan információ erőforrásra, amely az előbbi leírását szolgáltatja. HTTP válaszokban a 303-as állapotkód jelzi az átirányítást, amelyhez a `Location` fejlécmező tartalmazza azt az URI-t, amelyre az átirányítás történik.

A megoldás kombinálható tartalom-egyeztetéssel. Így működik például a DBpedia [5], amely minden erőforráshoz három különböző URI-t biztosít az alábbi minta szerint:

- `http://dbpedia.org/resource/Bart_Simpson` egy olyan URI, amely egy nem információ erőforrást azonosít
- `http://dbpedia.org/data/Bart_Simpson` a nem információs erőforrás RDF/XML reprezentációját szolgáltató információs erőforrás URI-ja
- `http://dbpedia.org/page/Bart_Simpson` a nem információs erőforrás HTML reprezentációját szolgáltató információs erőforrás URI-ja

Nem információ erőforrást azonosító URI hivatkozás-feloldásakor a kliens számára megfelelőbb információs erőforráshoz történik átirányítás.

6.3. „Kapcsolt adatok" szolgáltatása

Kézenfekvő „kapcsolt adatok" szolgáltatása olyan statikus RDF/XML dokumentumokban, amelyekben az erőforrásokat a 6.2.1. részben bemutatott hash URI-k azonosítják. Sajnos ez az út sok esetben járhatatlan. Akár egyetlen hash URI hivatkozás-feloldása a teljes dokumentum reprezentációként

továbbítását eredményezi, amely nagyméretű állományoknál nemkívánatos hálózati terhelést jelenthet. (A kliens oldal természetesen használhat a hatékonyabb működéshez gyorstárat.)

A létező adatok többségét egyébként sem RDF/XML dokumentumok tárolják, hanem például adatbázisok, amelyek teljes tartalma elméletileg exportálható ugyan RDF/XML dokumentumokba, de ez megoldásként a legtöbb esetben elfogadhatatlan.

A D2R Server [34] és a Triplify [26] relációs adatbázisok tartalmát kínálják „kapcsolt adatokként". Az előbbi az adatbázis-séma leképezését teszi lehetővé egy adott RDF szókészletre, de képes alkalmas szókészletet és a megfelelő leképezést automatikusan előállítani, ráadásul biztosít SPARQL végpontot is. Az utóbbi pedig egy olyan webalkalmazás, amely „kapcsolt adatok" formájában szolgáltatja előre rögzített SQL lekérdezések eredményét, és amelyhez több népszerű webalkalmazáshoz (mint például a Drupal, phpBB és Joomla!) adottak kész konfigurációk. A két eszköz szabad és nyílt forrású.

Sok Web 2.0 webhely biztosít webszolgáltatásokat, amelyekkel strukturált adatokként – többnyire XML vagy JSON [40] formátumban – érhetők el bizonyos tartalmak. A gyakorlatban REST-típusú webszolgáltatások [52] alkalmazása elterjedt. Csupán néhány példa: Amazon Web Services [1], Google Data Protocol [9], Flickr Services [8], Last.fm Web Services [12].

Webszolgáltatásokhoz megvalósítható olyan úgynevezett **Linked Data wrapper**, amely alkalmas URI-t rendel minden olyan erőforráshoz, amelyről a szolgáltatás adatokat biztosít. Ez URI hivatkozás-feloldás során a kliens számára transzparens módon szólítja meg a webszolgáltatást, az eredményt pedig alkalmas formába alakítva továbbítja reprezentációként. Így szolgáltat „kapcsolt adatokat" például a Last.fm RDFizer [13] és az RDF Book Mashup [23].

Egyéb adatforrások esetében megoldást jelenthet úgynevezett **RDFizer** [24] eszközök használata, amelyek az információtartalom RDF kijelentések formájában történő kinyerésérére képesek. Nagytömegű RDF kijelentést adatbázisban célszerű tárolni, amelyet számos eszköz lehetővé tesz. Megfelelő a célra például a szabad és nyílt forrású Jena Semantic Web Framework [11] SDB és TDB komponense, amelyek közül az előbbi relációs adatbáziskezelő rendszerben tárolja az adatokat, az utóbbi pedig egy natív RDF adatbázist valósít meg. RDF tároló eleve biztosíthat Linked Data interfészt. Ha csupán egy SPARQL végpont áll rendelkezésre, akkor például a Pubby Linked Data Frontend [22] kínál megoldást.

[35] egy alapos áttekintését adja „kapcsolt adatok" szolgáltatásához rendelkezésre álló módszereknek és eszközöknek.

6.4. Linked Data adathalmazok

A **Linking Open Data (LOD)** [14] projekt ernyője alatt számos szabadon felhasználható Linked Data adathalmaz található, amelyek értékét az is növeli, hogy RDF linkekkel kapcsolódnak egymáshoz. Közülük több nem csupán URI hivatkozás-feloldás révén érhető el, hanem letölthető állományok formájában is, az RDF valamely szintaxisával ábrázolva. Sok esetben a lekérdezéshez rendelkezésre áll SPARQL végpont.

A LOD projekt 2007-ben indult az „adat web" megteremtésének céljával. Keretében szabadon rendelkezésre álló adathalmazokat konvertáltak RDF-be és tettek elérhetővé „kapcsolt adatok" formájában. Míg az indulásnál elsősorban kutatók bábáskodtak a projekt körül, később számos további projekt, szervezet és intézmény csatlakozott a kezdeményezéshez.

A rendelkezésre álló adathalmazokat egy helyen összegyűjtve a Comprehensive Knowledge Archive Network (CKAN) LOD csoportjában [2] lehet elérni, amely az adathalmazokról metaadatokat is szolgáltat. Az oldalon jelenleg kétszáznál több adathalmazt találunk.

A LOD adathalmazokról [36] közöl statisztikákat. Az összes tartalmazott RDF kijelentés száma eszerint meghaladja a 25 milliárdot, az adathalmazok között van olyan – az alább tárgyalt Data.gov adathalmaz –, amely ehhez önmagában 6 milliárdnál több RDF hármassal járul hozzá. Közel 400 millió RDF link kapcsol össze továbbá különböző adathalmazokat.[3] A 6.1. ábra a LOD adathalmazokat és kapcsolataikat mutatja, amelyen az élek az adathalmazok erőforrásai közötti RDF linkekre utalnak.

A LOD adathalmazok változatos témájúak és számos különböző tudásterületet ölelnek fel. A szerző személyes kedvence a DBpedia projektben [5, 37] létrehozott és a Wikipédia tartalmát kínáló adathalmaz. Olyan nagyon hatékonyan kiaknázható információforrás ez, amelyet alkalmazások széles köre hasznosíthat.[4]

Trend ma a Linked Data adatszolgáltatók közé felsorakozni, az elképzelés különösen népszerű a könyvtáros közösségben – lásd például a washingtoni Kongresszusi Könyvtár [15] vagy a Német Nemzeti Könyvtár [47] Linked Data szolgáltatását –, de példaértékű a New York Times szolgáltatása [18] is.

Egy hazai szolgáltató az Országos Széchényi Könyvtár, amely 2010 áprilisában tette elérhetővé online katalógusának és Digitális Könyvtárának teljes tartalmát Linked Data formájában [20]. Ehhez olyan közismert RDF szókészleteket használnak, mint például a Dublic Core [6, 7], FOAF [39],

[3] A számok a 2010 októberében aktuális állapotot tükrözik.

[4] A DBpedia lehetséges felhasználásainak és létező alkalmazásainak számbavételét a projekt weboldalán [5] találjuk a *Use Cases* és *Applications* hiperhivatkozások alatt.

SIOC [33] és SKOS [51]. A lekérdezéshez SPARQL végpontot biztosítanak. Az adathalmaz a LOD projekt része, RDF linkekkel kapcsolódik a DBpedia adathalmazhoz.

2009 májusában indult az Egyesült Államokban a Data.gov [3] kormányzati portál, amely szövetségi kormányzati adatokat tesz elérhetővé gépi feldolgozásra alkalmas formában. A kezdeményezés céljai között szerepel az átláthatóság és nyitottság növelése a kormányzásban, ezáltal a demokrácia erősítése, a kormányzati munka hatékonyságának növelése.

Még ugyanebben az évben RDF-be konvertálták a Data-gov Wiki [4, 43] projekt keretében az aktuálisan rendelkezésre álló Data.gov adathalmazokat, létrehozva így a mai napig legnagyobb LOD adathalmazt.

Időközben a Data.gov portálon szabad felhasználásra kínált adathalmazok száma alig egy év alatt 47-ről 270 000-re nőtt, amelyek felbecsülhetetlen értéket jelentenek. Ma a Data.gov portál már nem csupán a korábbi formátumokban (például CSV, XML és XLS) kínálja letöltésre az adathalmazokat, hanem a Data.gov Wiki projektnek otthont adó intézménnyel együttműködve RDF/XML-ben is.[5]

6.5. A szemantikus web megvalósulása?

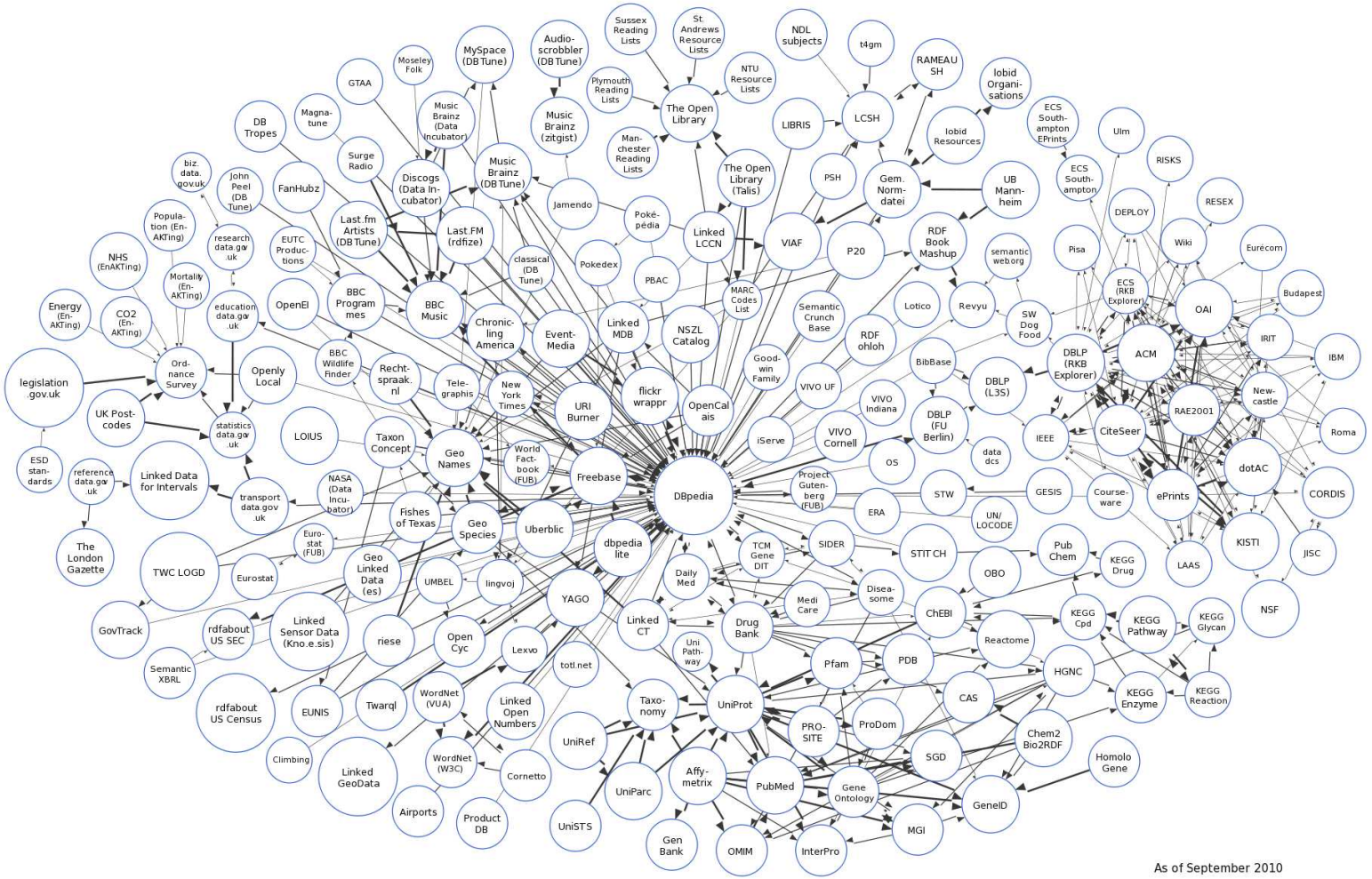
A Linked Data elképzelés és a webszolgáltatások is strukturált adatok szabványos módon elérhetővé tételét célozzák meg, azonban a Linked Data egy olyan egyszerű és univerzális megoldást jelent, amely nem igényel alkalmazásfüggő interfészeket vagy adatformátumokat, ráadásul transzparens módon teszi lehetővé különböző adatforrások integrálását.

Tim Berners-Lee, a web és a szemantikus web atyja az „adat webet” a szemantikus web megvalósulásának nevezi.[6] A nézetet nem mindenki osztja, a kijelentés vitákat generált a szemantikus web közösségen belül. Egy megfontolandó érveket és ellenérveket is felsorakoztató párbeszéd követhető nyomon a [49] blogbejegyzés kapcsán.

Az vitathatatlan, hogy a LOD adathalmazok alkalmas terepet kínálnak szemantikus web alkalmazások megvalósításához. A kiaknázás már megkezdődött, számos, az adathalmazokra épülő innovatív alkalmazás létezik. Említhető akár a Wikipedia keresést új alapokra helyező Faceted Wikipedia Search [46] szolgáltatás vagy a mobil eszközökre az aktuális GPS pozíciónak megfelelő Wikipedia tartalmat szolgáltató DBpedia Mobile [27]. További alkalmazásokat sorol fel [48] és [14].

[5] Jelenleg sajnos csak a letölthető adathalmazok egy része böngészhető is online.

[6] Például a [30] előadásban hangzott el a Linked Data elképzelés kapcsán a „Semantic web done right” kijelentés.



As of September 2010

94 6.1. ábra. Linking Open Data „felhő”: a projekt keretében elérhető adathalmazok (az ábra Richard Cyganiak honlapjáról [41] származik)

Irodalomjegyzék

[1] Amazon Web Services. URL http://aws.amazon.com/.

[2] CKAN LOD Group. URL http://ckan.net/group/lodcloud.

[3] Data.gov. URL http://www.data.gov/.

[4] Data-gov Wiki. URL http://data-gov.tw.rpi.edu/wiki.

[5] DBpedia. URL http://dbpedia.org/.

[6] Dublin Core Metadata Element Set. DCMI Recommendation, 2008. URL http://dublincore.org/documents/dces/. version 1.1.

[7] DCMI Metadata Terms. DCMI Recommendation, 2008. URL http://dublincore.org/documents/dcmi-terms/.

[8] Flickr Services. URL http://www.flickr.com/services/api/.

[9] Google Data Protocol. URL http://code.google.com/apis/gdata/.

[10] Apache HTTP Server. URL http://httpd.apache.org/.

[11] Jena Semantic Web Framework. URL http://jena.sourceforge.net/.

[12] Last.fm Web Services. URL http://www.last.fm/api.

[13] Last.fm RDFizer. URL http://lastfm.rdfize.com/.

[14] Linking Open Data. URL http://esw.w3.org/SweoIG/TaskForces/CommunityProjects/LinkingOpenData.

[15] Library of Congress Authorities and Vocabularies. URL http://id.loc.gov/.

[16] Marbles Linked Data Engine. URL http://marbles.sourceforge.net/.

[17] MIME Media Types. URL http://www.iana.org/assignments/media-types/.

[18] New York Times – Linked Open Data. URL http://data.nytimes.com/.

[19] OpenLink Data Explorer Extension. URL http://ode.openlinksw.com/.

[20] Az Országos Széchényi Könyvtár a szemantikus weben. URL http://nektar.oszk.hu/wiki/Szemantikus_web.

[21] Web Ontology Language (OWL). URL http://www.w3.org/2004/OWL/.

[22] Pubby linked data frontend. URL http://www4.wiwiss.fu-berlin.de/pubby/.

[23] RDF Book Mashup. URL http://www4.wiwiss.fu-berlin.de/bizer/bookmashup/.

[24] RDFizers. URL http://simile.mit.edu/wiki/RDFizers.

[25] Tabulator: Generic data browser. URL http://www.w3.org/2005/ajar/tab.

[26] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, and David Aumueller. Triplify – light-weight linked data publication from relational databases. In *Proceedings of Semantic Data Web Track of 18th International World Wide Web Conference (WWW 2009)*, pages 621–630. ACM, 2009.

[27] Christian Becker and Chris Bizer. DBpedia Mobile: A Location-Enabled Linked Data Browser. In *Linked Data on the Web (LDOW2008)*, 2008. URL http://beckr.org/wp-content/uploads/DBpediaMobile.pdf.

[28] Tim Berners-Lee. Cool URIs don't change, 1998. URL http://www.w3.org/Provider/Style/URI.html.

[29] Tim Berners-Lee. Linked Data, 2006. URL http://www.w3.org/DesignIssues/LinkedData.html.

[30] Tim Berners-Lee. Linked Open Data, 2008. URL http://www.w3.org/2008/Talks/0617-lod-tbl/. Talk at Linked Data Planet.

[31] Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and Analyzing linked data on the Semantic Web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006.

[32] Diego Berrueta and Jon Phipps. Best Practice Recipes for Publishing RDF Vocabularies. W3C Working Group Note, 2008. URL `http://www.w3.org/TR/swbp-vocab-pub/`.

[33] Diego Berrueta, Dan Brickley, Stefan Decker, Sergio Fernández, Christoph Görn, Andreas Harth, Tom Heath, Kingsley Idehen, Kjetil Kjernsmo, Alistair Miles, Alexandre Passan, Axel Polleres, and Luis Polo. SIOC Core Ontology Specification, 2010. URL `http://rdfs.org/sioc/spec/`.

[34] Chris Bizer and Richard Cyganiak. D2R Server – Publishing Relational Databases on the Semantic Web. URL `http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/`.

[35] Chris Bizer, Richard Cyganiak, and Tom Heath. How to Publish Linked Data on the Web. URL `http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/`.

[36] Chris Bizer, Anja Jentzsch, and Richard Cyganiak. State of the LOD Cloud. URL `http://www4.wiwiss.fu-berlin.de/lodcloud/state/`.

[37] Christian Bizer, Jens Lehmann, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia – a crystallization point for the web of data. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, (7):154–165, 2009.

[38] Dan Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004. URL `http://www.w3.org/TR/rdf-schema/`.

[39] Dan Brickley and Libby Miller. FOAF Vocabulary Specification, 2010. URL `http://xmlns.com/foaf/spec/`. version 0.97.

[40] D. Crockford. The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational), 2006. URL `http://www.ietf.org/rfc/rfc4627.txt`.

[41] Richard Cyganiak. The Linking Open Data cloud diagram. URL http://richard.cyganiak.de/2007/10/lod/.

[42] Steve DeRose, Eve Maler, David Orchard, and Norman Walsh. XML Linking Language (XLink) Version 1.1. W3C Recommendation, 2004. URL http://www.w3.org/TR/xlink11/.

[43] Li Ding, Dominic DiFranzo, Sarah Magidson, Deborah L. McGuinness, and Jim Hendler. Data-Gov Wiki: Towards Linked Government Data, 2010. URL http://data-gov.tw.rpi.edu/2010/linkedai-2010-datagov.pdf.

[44] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Standard), 1999. URL http://www.ietf.org/rfc/rfc2616.txt.

[45] Tim Finin, Yun Peng, R. Scott Cost, Joel Sachs, Anupam Joshi, Pavan Reddivari, Rong Pan, Vishal Doshi, and Li Ding. Swoogle: A search and metadata engine for the semantic web. In *In Proceedings of the Thirteenth ACM Conference on Information and Knowledge Management*, pages 652–659. ACM, 2004.

[46] Rasmus Hahn, Christian Bizer, Christopher Sahnwaldt, Christian Herta, Scott Robinson, Michaela Bürgle, Holger Düwiger, and Ulrich Scheel. Wikipedia Faceted Search. In *Business Information Systems*, volume 47 of *Lecture Notes in Business Information Processing*, pages 1–11. Springer, 2010.

[47] Jan Hannemann and Jürgen Kett. Linked Data for Libraries. In *Proceedings of the World Library and Information Congress: 76th IFLA General Conference and Assembly*, 2010.

[48] Michael Hausenblas. Linked Data Applications—The Genesis and the Challenges of Using Linked Data on the Web. Technical report, Digital Enterprise Research Institute (DERI), 2009.

[49] Tom Heath. Linked Data? Web of Data? Semantic Web? WTF?, 2009. URL http://tomheath.com/blog/2009/03/linked-data-web-of-data-semantic-web-wtf/. Blog entry.

[50] Graham Klyne and Jeremy J. Carroll. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, 2004. URL http://www.w3.org/TR/rdf-concepts/.

[51] Alistair Miles and Sean Bechhofer. SKOS Simple Knowledge Organization System Reference. W3C Recommendation, 2009. URL `http://www.w3.org/TR/skos-reference/`.

[52] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2007. ISBN 978-0-596-80168-7.

[53] Leo Sauermann and Richard Cyganiak. Cool URIs for the Semantic Web. W3C Interest Group Note, 2008. URL `http://www.w3.org/TR/cooluris/`.

[54] Giovanni Tummarello, Renaud Delbru, and Eyal Oren. Sindice.com: Weaving the Open Linked Data. In *Proceedings of the International Semantic Web Conference (ISWC)*, pages 552–565, 2007. URL `http://www.eyaloren.org/pubs/iswc2007.pdf`.

7. fejezet

Csomag metaadatok publikálása

7.1. Bevezetés

Noha sok csomagkezelési megoldás létezik, a csomagok egy közös jellemzője, hogy sok metaadatot hordoznak. Mivel a szerző egyaránt lelkes híve a Linux-módra történő csomagkezelésnek és a szemantikus webnek, elég nyilvánvaló volt számára a következő feladat: tegyünk elérhetővé szoftvercsomag metaadatokat szemantikus web alkalmazások számára is!

Az ebben a fejezetben bemutatásra kerülő munka részeként a szerző olyan eszközöket fejlesztett ki, amelyek csomagokból metaadatokat nyernek ki és RDF-be alakítják az információkat. Az eszközök speciális RDF szókészleteket használnak csomag metaadatok ábrázolásához. A munka eredményeként több OWL webontológia készült, amelyek a támogatott csomagformátumokhoz definiálják a szókészleteket.

Az előbbi programokat egy-egy állományhoz lehet használni, nem alkalmasak csomagok közötti függőségek és egyéb kapcsolatok interaktív módon történő vizsgálatához. A konverziós eszközök működésének egy logikus továbbgondolása a csomag metaadatok „kapcsolt adatok" formájában történő szolgáltatása, amelyre egy megoldást ad a szerző.

A munkát a szerző korábbi, RPM csomagokat feldolgozó RDF kinyerő és a 3. fejezetben ismertetett programja ihlette, annak „újraélesztése" történt a megvalósítás során.

A 7.2. szakasz az RDF szoftvercsomagok kezeléséhez történő gyakorlati felhasználásait tekinti át. A 7.3. szakasz az egyedi csomagokból metaadatokat kinyerő konverziós programokat tárgyalja. A 7.4. szakasz témája a csomagok OWL-ben történő modellezése, a következő 7.5. szakasz pedig ennek gya-

korlati felhasználásaként egy Linked Data szolgáltatás megvalósítására ad példát.

7.2. RDF és szoftvercsomagok

Az RDF egy egyszerű és univerzális megoldást ad metaadatok ábrázolásához, azonban viszonylag kevés gyakorlati felhasználása létezik csomagok kezeléséhez. Ezek közül néhányat sorolunk fel alább.

7.2.1. rpmfind.net

Az rpmfind.net [7] egy számos RPM-alapú Linux-disztribúció csomagjait katalogizáló kereshető adatbázis, amely számára [17] egy RDF-alapú megoldást javasol a csomag metaadatok tárolásához.[1]

7.2.2. GNUpdate

A GNUpdate [2] egy szabad és nyílt forrású, Linux rendszerekhez rendelkezésre álló univerzális csomagkezelő rendszer volt, amely a csomagok metaadatait az RDF XML szintaxisával (RDF/XML) ábrázolva tárolta. Fejlesztése azonban félbeszakadt, a projekt honlapján 2004 óta nem észlelhető aktivitás.

7.2.3. XPInstall

Az XPInstall [10] a Mozilla-alapú platformokon kiterjesztések telepítését megvalósító platformfüggetlen csomagkezelési megoldás. Az XPI telepítő moduloknak nevezett csomagok egy `install.rdf` nevű állományban az RDF XML szintaxisával (RDF/XML) ábrázolva tartalmaznak metaadatokat.

7.2.4. SPDX

2010-ben jelentette be a Linux Foundation a Software Package Data Exchange (SPDX) [8] specifikációt, amelyet kiemelt fontosságúnak tekint. Az SPDX célja, hogy szabványos, automatikus feldolgozásra alkalmas formában tegye lehetővé szoftvercsomagokkal kapcsolatos szerzői jogi információk és felhasználási feltételek leírását. A metaadatok ábrázolásához az RDF XML

[1] Egy időben bizonyosan ezt a megoldást használták, azonban a csomag metaadatokat RDF-ben ma már nem lehet elérni. Sajnos nem áll rendelkezésre nyilvános információ az adattárolás jelenlegi megvalósításáról.

szintaxisát (RDF/XML) használja. Támogatja a teljes csomagra vonatkozó áttekintő információk tárolását (például név, rövid és hosszú leírás), valamint a tartalmazott állományok jellemzését is (például elérési útvonal, típus). Az elterjedten használt szoftverlicencekhez olyan azonosítókat rendel, amelyek segítségével egységes módon lehet ezekre hivatkozni. Az SPDX elsődlegesen a szerzői jogokra és felhasználási feltételekre helyezi a hangsúlyt, például a szoftvercsomagok közötti függőségek ábrázolásához nem biztosít támogatást.

7.3. RDF kinyerés egyedi csomagokból

A 3. fejezetben került bemutatásra a szerző által kifejlesztett RPM csomagokat feldolgozó RDF kinyerő program. A korábbi munka folytatásaként két további csomagformátumhoz is készített a szerző hasonló konverziós programokat: Debian csomagokhoz és az R statisztikai környezet által használt csomagokhoz. Az előbbiekhez ugyan egy másik fejlesztő korábban már közreadott az RDFizers [5] projektben egy RDF kinyerőt, a szerző a csomag metaadatok ábrázolásához használt RDF szókészletekkel történő kísérletezéséhez hasznosabbnak találta a saját megvalósítását.

A Debian és az R csomagok közönséges archív állományok: előbbiek a Unix-szerű rendszerekben használt `ar` archív állományok, utóbbiak pedig ZIP állományok vagy a `gzip` programmal tömörített `tar` archívumok.[2] A legfontosabb metaadatokat mindkét esetben egy-egy olyan vezérlő állománynak nevezett szövegállomány tartalmazza, amelynek szerkezetét a 7.1. ábra mutatja. A vezérlő állományokat mezőknek nevezett név-érték párok alkotják.

A vezérlő állományok kinyerése és feldolgozása egyszerű feladat, ezért eltekintünk a konverziós programok működésének részletes tárgyalásától. A megvalósítás Java programozási nyelven történt, az archívumok tartalmához a szerző a szabad és nyílt forrású `Commons Compress` [1] osztálykönyvtárral fért hozzá, így a programok valóban platformfüggetlenek, használatukhoz nincs szükség Unix-szerű környezetekhez kötődő rendszereszközök rendelkezésre állására.

Az eszközök szolgáltatása elérhető REST-stílusú webszolgáltatásokkal [13] is, amelyek egy csomag URI-ját kapják meg és RDF-ben adják vissza a metaadatokat. A webszolgáltatások a Restlet [6] keretrendszerben kerültek implementálásra.

[2]Debian csomagok esetében ez a legtöbb felhasználó által használt bináris csomagokra igaz, a forráscsomagokat több állomány alkotja.

```
Package: file
Version: 4.26-1
Architecture: amd64
Maintainer: Daniel Baumann <daniel@debian.org>
Installed-Size: 140
Depends: libc6 (>= 2.7-1), libmagic1 (= 4.26-1), zlib1g (>= 1:1.1.4)
Section: utils
Priority: standard
Homepage: http://www.darwinsys.com/file/
Description: Determines file type using "magic" numbers
 File tests each argument in an attempt to classify it. There are three sets of
 tests, performed in this order: filesystem tests, magic number tests, and
 language tests. The first test that succeeds causes the file type to be
 printed.
 .
 Starting with version 4, the file command is not much more than a wrapper
 around the "magic" library.
```

7.1. ábra. Debian csomag vezérlő állománya

7.4. Csomagok modellezése

7.4.1. A modellezési feladat kihívásai

Látni fogjuk, hogy nem is olyan egyszerű feladat szoftvercsomagok modellezéséhez alkalmas OWL webontológia megalkotása, amely felhasználható Linked Data szolgáltatás megvalósításához is.

Az egyik alapvető nehézséget a csomagok között megadható kapcsolatok jelentik, amelyek az alábbi kihívások elé állítanak:

- A csomagok között definiálható kapcsolatok sokfélesége: például a Debian rendszerben nagyjából tízféle különböző kapcsolat használható.
- A csomagok között áttételesen definiálható kapcsolatok: a csomagok nem csupán csomagoktól függhetnek, hanem olyan funkcióktól is, amelyeket akár több csomag is nyújthat.
- A csomagok közötti kapcsolatokban korlátozható a verziószám, például megadható minimális verziószám.

Egy más jellegű probléma abból ered, hogy egy szoftvercsomagnak sok különböző verziója létezhet egyidejűleg:

- Azonos nevű, de különböző verziószámú csomagokra ugyanazon absztrakt entitás (például egy alkalmazás) megtestesüléseiként tekinthetünk.
- A metaadatok mindig az egyes verziókhoz állnak rendelkezésre, azonban a mögöttes absztrakt entitásról csak ezeken keresztül kaphatunk információkat.

A továbbiakban a fenti absztrakt entitásokra **absztrakt csomag** néven hivatkozunk.

7.4.2. Csomagok kapcsolatai

A modellezési feladat bemutatásához röviden áttekintjük a csomagok közötti kapcsolatok kezelésének néhány gyakorlati megvalósítását. Három különböző csomagformátumot vizsgálunk, kellően reprezentatív képet adva az elterjedten használt megoldásokról.

7.4.2.1. R

Az R statisztikai és grafikai környezet négy különböző csomagok közötti kapcsolatot kezel [15]:

- `Depends`/`Imports`: mindkettő azt jelenti, hogy a kapcsolatot deklaráló csomaghoz az adott csomagok szükségesek[3]
- `Enhances`: azt jelenti, hogy a kapcsolatot deklaráló csomag „hozzáadott értéket" tartalmaz a felsorolt csomagokhoz
- `Suggests`: azt jelenti, hogy a kapcsolatot deklaráló csomaghoz ajánlott, de nem kötelező az adott csomagok rendelkezésre állása

A kapcsolatok megadása a vezérlő állományokban azonos nevű mezőkkel lehetséges. A kapcsolatokban a csomagokhoz mind a négy esetben előírható a `<=`vagy `>=` operátort tartalmazó verziószám korlátozás.

Egy kapcsolódó további mező a `SystemRequirements`, amely a csomagkezelő rendszer hatáskörén kívül eső követelmények kifejezésére szolgál. Ilyen követelmény például bizonyos programkönyvtárak rendelkezésre állása az operációs rendszerben.

Néhány tipikus példa a kapcsolatok a csomagok vezérlő állományában történő ábrázolására:

```
Depends: stats, gnm (>= 1.0-0), colorspace
Enhances: tm (>= 0.5)
Imports: digest, stringr (>= 0.4), mutatr, evaluate (>= 0.3)
Suggests: Biobase (>= 2.5.5), statmod
SystemRequirements: libpng
```

[3] A különbség a kettő között technikai.

7.4.3. Debian

A Debian csomagok között kialakítható kapcsolatok részletes tárgyalása meghaladja a dolgozat kereteit, a hivatalos dokumentáció [16] egy teljes fejezetet szentel a témának.

Bináris csomagokhoz a következő kapcsolatok alkalmazhatók:

- `Breaks`/`Conflicts`: csomagok összeférhetetlenségét kifejező kapcsolatok
- `Depends`/`Pre-Depends`: abszolút függést kifejező kapcsolatok a deklaráló csomaghoz elengedhetetlenül szükséges csomagok megadásához
- `Enhances`: azt kifejező kapcsolat, hogy a deklaráló csomag „hozzáadott értéket" tartalmaz a felsorolt csomagokhoz
- `Recommends`/`Suggests`: ajánlást kifejező kapcsolatok
- `Replaces`: kettős jelentésű kapcsolat, amely azt jelzi, hogy a kapcsolatot deklaráló csomag felülírhat állományokat az adott csomagokból, vagy azt, hogy a deklaráló csomag telepítése során az adott csomagokat összeférhetetlenség esetén el kell távolítani

Csak forráscsomagokhoz használható további kapcsolatok a `Build-Depends`, `Build-Depends-Indep`, `Build-Conflicts` és `Build-Conflicts-Indep`. A kapcsolatok megadása a vezérlő állományokban azonos nevű mezőkkel történik.

Több csomag nyújthatja nagyjából ugyanazt a funkcionalitást, ennek kezeléséhez biztosítja a csomagkezelő rendszer a **virtuális csomagok**at. Egy virtuális csomag mindössze egy olyan funkcionalitást reprezentáló név, amelyet tipikusan több csomag is szolgáltat. A csomagok a `Provides` mező segítségével jelezhetik, hogy bizonyos virtuális csomagoknak megfelelő funkciókat nyújtanak, mint például:

```
Provides: ftp-server
```

Ilyenkor azt mondjuk, hogy a csomag a mezőben adott virtuális csomago(ka)t szolgáltatja. Virtuális csomagok nevei megjelenhetnek kapcsolatokban, ezek helyére behelyettesíthető bármely a virtuális csomagot szolgáltató csomag.

A `Provides` kivételével minden mezőben tartozhat a csomagokhoz egy olyan verziószám korlátozás, amelyben a következő relációs operátorok állnak rendelkezésre: `<<` (szigorúan kisebb), `<=` (kisebb vagy egyenlő), `=` (egyenlő), `>=` (nagyobb vagy egyenlő), `>>` (szigorúan nagyobb). Néhány példa a kapcsolatok ábrázolására:

```
Depends: libc6 (>= 2.7-1), perl
Suggests: sun-java6-demo, openjdk-6-doc, sun-java6-source
Breaks: xserver-xorg-core (<< 2:1.6)
```

Bizonyos mezőkben alternatívákat is meg lehet adni | karakterekkel elválasztva, mint például:

```
Depends: perl, curl | lynx
```

Csoportosításnál a | karakter nagyobb precedenciájú a vessző karakternél, tehát a fenti sor azt jelenti, hogy a `perl`, és a `curl` vagy `lynx` csomag szükséges.

Forráscsomagok kapcsolataiban megjelenő minden egyes csomaghoz tartozhat olyan architektúra korlátozás, amelynek szintaxisát az alábbi példa szemlélteti:

```
Build-Depends: valgrind [amd64 i386 powerpc]
```

A fenti sor azt jelenti, hogy a `valgrind` csomagot a kapcsolatban csak `amd64`, `i386` és `powerpc` architektúrák esetén kell figyelembe venni. Negációt jelöl a ! karakter, tehát például a

```
Build-Depends: libblas-dev [!arm !m68k]
```

kapcsolatban a `libblas-dev` csomagot csak az `arm` és `m68k` architektúrákon kell figyelmen kívül hagyni.

7.4.4. RPM

Az RPM formátum a csomagok közötti kapcsolatokat úgynevezett **képességek** (**capability**) segítségével valósítja meg, amelyek nagyjából a Debian rendszer virtuális csomagjainak felelnek meg. Míg azonban virtuális csomagok szolgáltatása a Debian rendszerben opcionális, addig minden RPM csomag számára kötelező a szolgáltatott képességek deklarálása.

Csomagok számára csupán négy lehetőséget biztosít a formátum kapcsolatok kezeléséhez:

- `CONFLICT`: a deklaráló csomaggal összeférhetetlen csomagok megadására szolgál
- `OBSOLETE`: a deklaráló csomag segítségével „elavultként” jelölhet meg csomagokat
- `PROVIDE`: a deklaráló csomag által nyújtott képességek megadására szolgál

- `REQUIRE`: a deklaráló csomaghoz elengedhetetlenül szükséges csomagok felsorolására szolgál

A formátum bináris, a metaadatok, így a kapcsolatok is az A. függelékben leírt fejléc struktúrában kerülnek tárolásra.

Minden képességet egy olyan karakterlánc azonosít, amely reprezentálhat egy állományt (például `/usr/bin/cancel`), egy absztrakt funkciót (például `webserver`) vagy csomagot (például `kernel`). A Debian virtuális csomagjaitól eltérően a szolgáltatott képességekhez tartozhat verziószám is. A kapcsolatokban képességek nevezhetők meg, amelyekhez megadható verziószám korlátozás a `<`, `<=`, `=`, `>=`, `>` operátorokkal.

Mivel az RPM csomagok bináris állományok, a szemléltetéshez célszerű azt megmutatni, hogy a csomagkezelő rendszer milyen formában jeleníti meg a kapcsolatokat a felhasználó számára. A `rpm` programmal az alábbi módon kérdezhető le, hogy az adott csomag milyen képességeket szolgáltat:

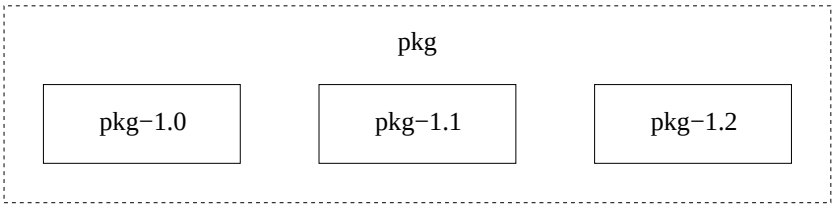
```
$ rpm -q --provides -p wget-1.12-2.fc13.x86_64.rpm
config(wget) = 1.12-2.fc13
webclient
wget = 1.12-2.fc13
wget(x86-64) = 1.12-2.fc13
```

Hasonlóképpen kapható meg az adott csomaghoz szükséges képességek listája:

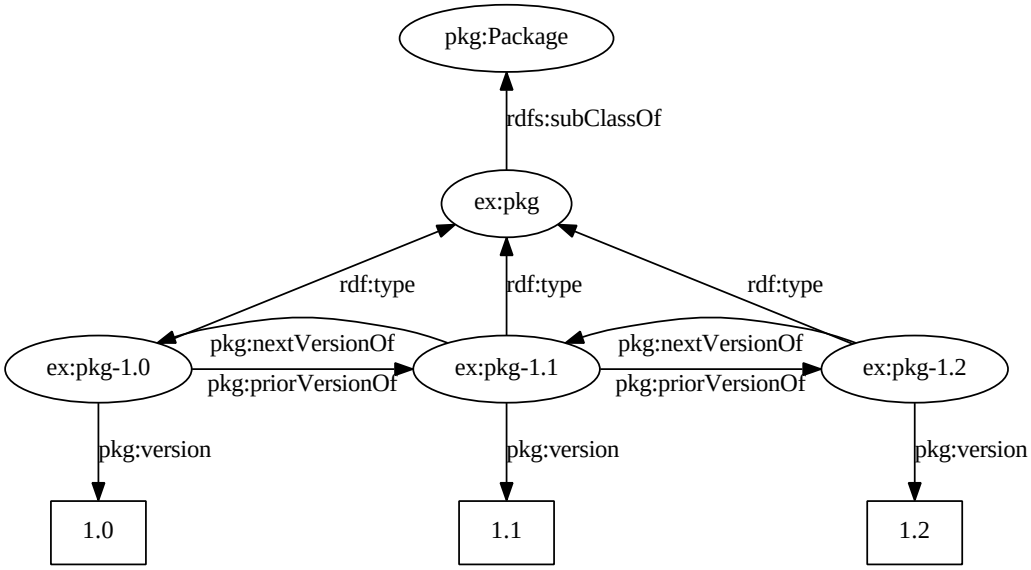
```
$ rpm -q --requires -p mc-4.7.1-2.fc13.x86_64.rpm
/bin/sh
/usr/bin/env
/usr/bin/perl
config(mc) = 1:4.7.1-2.fc13
dev >= 3.3-3
libc.so.6()(64bit)
libc.so.6(GLIBC_2.2.5)(64bit)
libc.so.6(GLIBC_2.3)(64bit)
...
```

7.4.5. Kapcsolatok ábrázolása

Különálló csomagokat kézenfekvő módon lehet RDF gráfokra leképezni, a kapcsolatokban megjelenő csomagneveket ábrázolhatják egyszerűen karakterlánc literálok. Így működik például a 3. fejezetben bemutatott RDF kinyerő program, amely RPM csomagokat dolgoz fel. Ez a megközelítés azonban teljesen alkalmatlan Linked Data szolgáltatás megvalósításához, mivel



(a) Csomag különböző verziói



(b) Csomag különböző verziói RDF gráffal ábrázolva

nem teszi lehetővé a csomagok közötti kapcsolatok RDF linkekkel történő navigálhatóságát. Gyakorlati felhasználhatóság szempontjából olyan ábrázolást célszerű tehát választani, amely Linked Data alkalmazások számára is megfelelő.

Egy jelentős lépést teszünk a helyes irányba, ha a modellbe bevezetjük az absztrakt csomagokat. A 7.2(a). ábrán ugyanannak a csomagnak három különböző verziója látható, amelyek összetartozását az azonos csomagnév fejezi ki. A 7.2(b). ábra mutatja a csomagok ábrázolásához a szerző által javasolt gráfszerkezetet.

Az ábra azt sugallja, hogy az absztrakt csomagokat osztályokként célszerű a modellbe felvenni. Az absztrakt csomagok osztályai egy, az összes csomagot reprezentáló osztály alosztályai. Az egyes csomagok a megfelelő absztrakt csomag osztályának példányaiként írhatók le, amely kifejező módon tükrözi a különböző verziók összetartozását.

A példában szereplő absztrakt csomag osztályát az alábbi módon lehet definiálni az OWL 2 funkcionális szintaxisának segítségével:

```
Declaration(Class(ex:pkg))
SubClassOf(ex:pkg pkg:Package)
SubClassOf(ex:pkg DataHasValue(pkg:name "pkg"^^xsd:string))
```

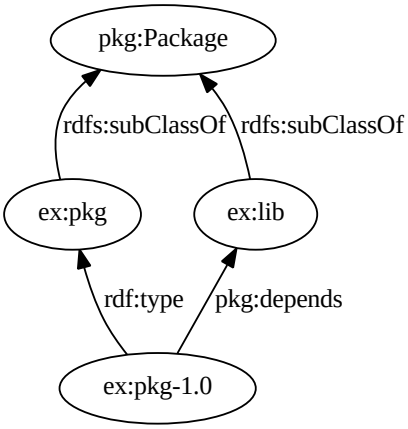
A fenti definíció biztosítja, hogy az egyes verziók neve rögzített (feltevés szerint a csomagnevet szolgáltató `pkg:name` tulajdonság funkcionális).

Adott csomaghoz tartozó különböző verziók sorrendiségét a 7.2(b). ábrán látható módon lehet leírni a `pkg:priorVersionOf` és `pkg:nextVersionOf` tulajdonságok segítségével.

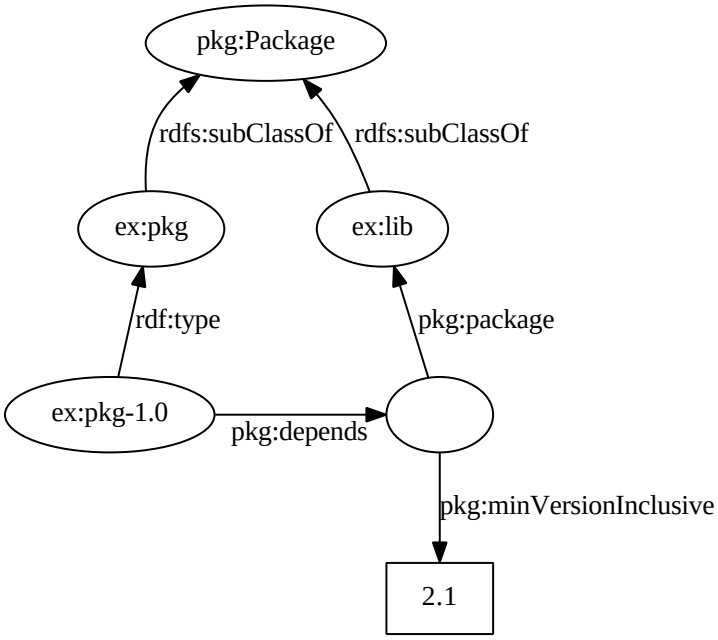
Vizsgáljuk most azt, hogy hogyan lehet a különböző csomagok közötti kapcsolatokat ábrázolni! Ez olyan csomagkezelő rendszerekben a legegyszerűbb, amelyek nem támogatják a kapcsolatokban megjelenő csomagokhoz verziószám korlátozások használatát. A lehetséges kapcsolatokat olyan tulajdonságok reprezentálják, amelynek értékeként az absztrakt csomagok osztályai megengedettek. A 7.2. ábra mutatja egy ilyen egyszerű modellben két csomag közötti függés leírását. A gráf azt fejezi ki, hogy a `pkg-1.0` csomaghoz a `lib` csomag valamely verziója szükséges.

Gyakorlati szempontból a fenti megoldás értéke csekély, mivel a csomagkezelő rendszerek általában megengedik a kapcsolatokban a verziószámok korlátozását. A RDF tulajdonságok binér relációkat képviselnek. Szükséges lehet azonban a kapcsolatok tulajdonságokkal jellemzése, amelyre az n-ér relációk RDF-beli modellezését tárgyaló [12] dokumentum ad megfelelő tervezési mintákat. A 7.3. ábrán látható, hogy a javasolt minta segítségével hogyan lehet a csomagok közötti kapcsolatokhoz a verziószámra vonatkozó korlátozást társítani. A kapcsolatot reprezentáló tulajdonság értékeként egy olyan üres csomópont jelenik meg, amely egy-egy tulajdonság segítségével hordozza a kapcsolat célpontját és az annak verziószámára vonatkozó korlátozást (lásd a `pkg:package` és `pkg:minVersionInclusive` tulajdonságokat). A gráf azt fejezi ki, hogy a `pkg-1.0` csomaghoz a `lib` csomag 2.1 vagy magasabb számú verziója szükséges.

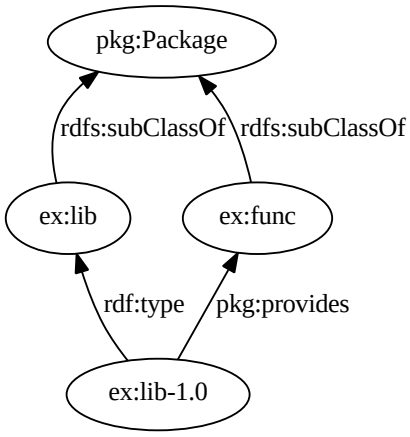
Virtuális csomagokhoz a szerző olyan osztályok definiálását ajánlja, mint az absztrakt csomagokhoz, így ezek a kapcsolatokban éppen olyan módon jelenhetnek meg, mint az absztrakt csomagok. Virtuális csomagok szolgáltatását azonban egy speciális tulajdonsággal célszerű jelezni, ahogy a 7.4. ábrán látható, amelyen megfigyelhető, hogy a virtuális csomagot szolgáltató csomag nem példánya a virtuális csomag osztályának. Utóbbi furcsának tűnhet, azonban indokolható. Eddig azzal a ki nem mondott feltételezéssel éltünk, hogy a virtuális és nem virtuális csomagok nevei különböznek. A Debian rendszerben megengedett azonban egyező nevű virtuális és nem virtuális csomagok használata. Ha ilyen kétértelmű csomagnév jelenik meg egy kapcsolatban, akkor a csomagkezelő rendszer az azonos nevű virtuális csomago-



7.2. ábra. Csomagok közötti függés ábrázolása (nincs verziószám korlátozás)



7.3. ábra. Csomagok közötti függés ábrázolása verziószám korlátozás használata esetén



7.4. ábra. Virtuális csomag szolgáltatása

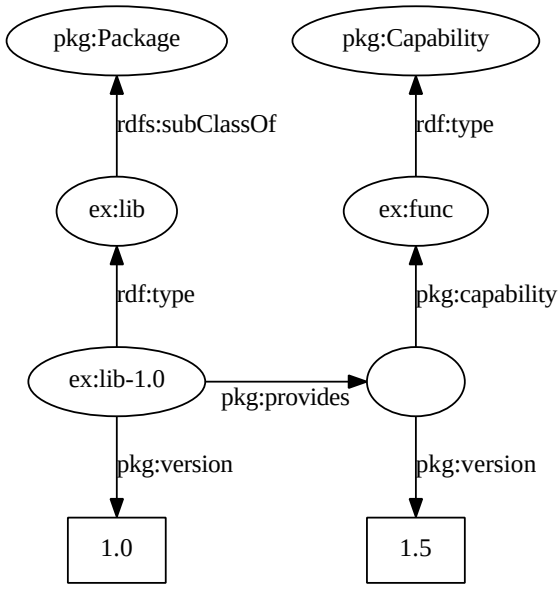
kat szolgáltató csomagokat és azonos nevű nem virtuális csomagokat is tekintheti helyette.[4] A probléma feloldásához a megegyező nevű nem virtuális absztrakt csomagokat és virtuális csomagokat ugyanaz az osztály kell hogy ábrázolja. Ebben az esetben le kell azonban mondani az osztály definiálásakor a csomagnév rögzítéséről (lásd a 109. oldalon bemutatott definíciót).

Némileg eltérő megoldást javasol a szerző a kapcsolatok ábrázolásához olyan csomagkezelő rendszerekben, ahol a csomagok szolgáltatott képességeken keresztül függenek egymástól, mint például az RPM formátum esetén. A 7.5(a). ábrán egy olyan RDF gráf látható, amely egy képesség egy csomag által történő szolgáltatását fejezi ki. A 7.5(b). ábra RDF gráfja ugyanennek a képességnek egy másik csomag által történő megkövetelését mutatja. A csomagokat a korábbi modellekben használt módon ábrázolják osztályok. Új elem azonban a képességeket reprezentáló osztály (az ábrán `pkg:Capability` néven látható). Emlékezzünk arra, hogy az RPM csomagok képességei karakterláncok formájában adottak, és hogy bármelyik csomag szolgáltathatja bármelyik képességet. Az előbbi osztály példányai felelnek meg ezeknek a képességeknek. A képességek szolgáltatásánál verziószám is megadható, így értelemszerűen verziószám korlátozás is megjelenhet a kapcsolatokban a képességekre történő hivatkozásoknál.

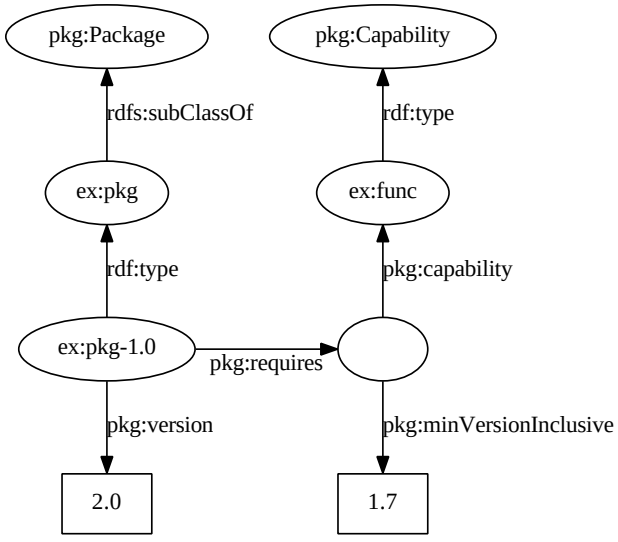
7.4.6. Webontológiák

A szerző az előző szakaszban bemutatott mintákat definiáló OWL webontológiákat dolgozott ki R, Debian és RPM csomagokhoz. A tervezés során

[4]Kivéve azt az esetet, ha meg van adva verziószám korlátozás, ekkor a virtuális csomagokat a csomagkezelő rendszer nem veszi figyelembe.



(a) Képesség csomag által történő szolgáltatása



(b) Csomag függése képességtől

elsődleges cél volt a csomagok közötti kapcsolatok RDF linkekkel történő közvetlen kifejezhetősége, amely biztosítja a felhasználhatóságot Linked Data alkalmazások megvalósításához.

A három csomagformátum közül az R statisztikai környezeté a legegyszerűbb, ehhez a csomagokat és kapcsolataikat tökéletesen modellezni képes webontológia készült. Ugyancsak teljes értékű megoldás született az RPM formátumhoz. Debian csomagokhoz sajnos azonban csak részleges megoldást sikerült adni. Az ontológia jelenleg csak bináris csomagok leírásához alkalmas, nem teszi lehetővé ugyanis architektúra korlátozások használatát, amelyek megengedettek forráscsomagok kapcsolataiban.

Az ábrázolás során információvesztés is történik: mindhárom formátumnál egy helyen történik a deklaráló csomaggal adott fajta kapcsolatban lévő csomagok felsorolása, ahol szerepe lehet a sorrendnek, amelyet a modellek nem képesek megőrizni.

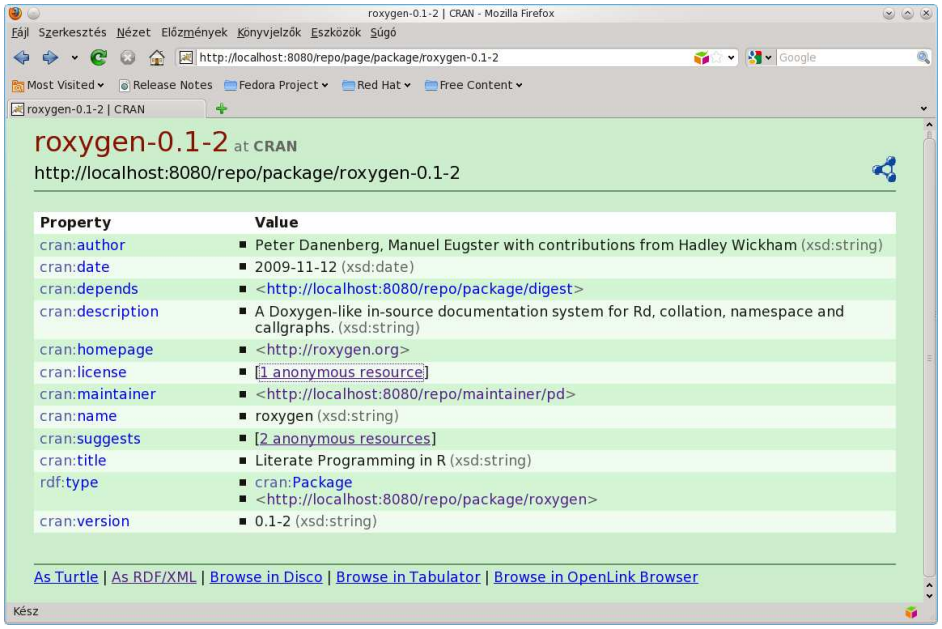
7.5. Linked Data szolgáltatás megvalósítása

Az ontológiák gyakorlati használatának bemutatásához a szerző egy megoldást adott R csomagtárolók metaadatainak „kapcsolt adatok" formájában történő publikálásához. Hasonló módon valósítható meg Linked Data szolgáltatás Debian és RPM csomagokhoz is.

Az R csomagtárolókat csomagokat tartalmazó olyan könyvtárak alkotják, amelyek mindegyikében megjelenik egy tartalomjegyzék szerepét betöltő és a könyvtárban lévő csomagok metaadatait összesítve kínáló állomány. Az állomány neve `PACKAGES`, a csomagok vezérlő állományainak összefűzésével kerül előállításra, benne az egyes csomagok rekordjait üres sorok választják el egymástól [14].[5]

Az egyedi R csomagokat feldolgozó RDF kinyerő program birtokában egyszerű volt megoldani a `PACKAGES` állományok tartalmának egyetlen RDF gráffá történő átalakítását. A konverzió után a szerző a szabad és nyílt forrású Joseki SPARQL szervert [3] használta SPARQL végpont biztosításához, amelyen keresztül lekérdezhetők az információk. A Joseki önmagában nem képes Linked Data szolgáltatásra, ehhez szükség volt még az ugyancsak szabad és nyílt forrású Pubby [4] eszközre, amely Linked Data interfészt nyújt SPARQL végpontokhoz.

[5]Egy `PACKAGES` állományban tipikusan csak a legfontosabb információk állnak rendelkezésre a csomagokról, Linked Data szolgáltatáshoz azonban olyan módon kell létrehozni, hogy minden mezőt tartalmazzon a vezérlő állományokból. Ezt az állományt előállító R függvény megfelelő paraméterezésével lehet elérni.



7.5. ábra. Csomag metaadatok böngészése

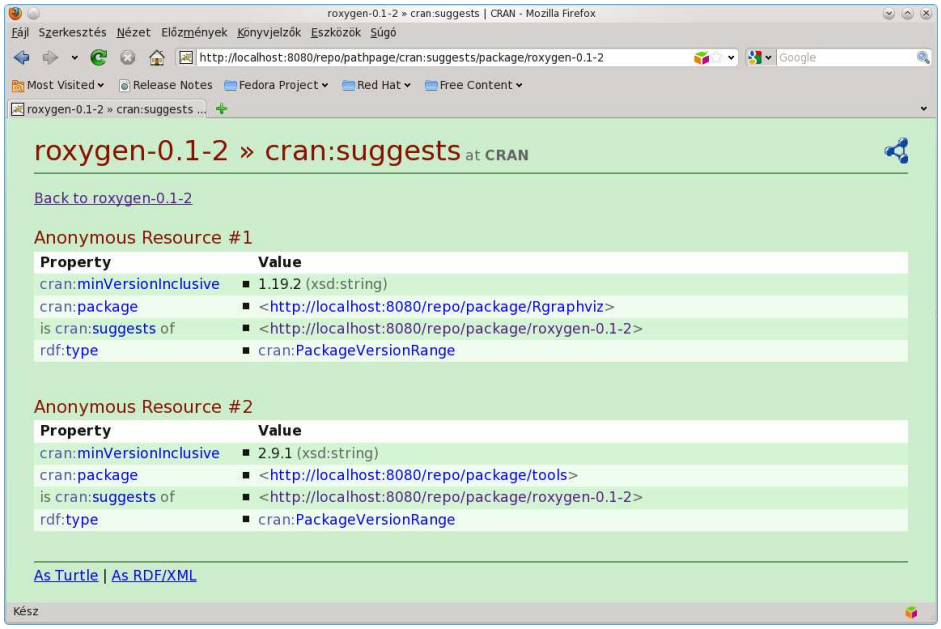
A 7.5. ábrán HTML nézetben láthatjuk a metaadatok böngészését. Megfigyelhető, hogy két tulajdonság értékeként üres csomópontok szerepelnek, ezek egy-egy kattintással elérhető leírásait mutatják a 7.6. és a 7.7. ábrák. A 7.8. ábrán a csomag karbantartója jelenik meg a FOAF [11] szókészlettel leírva. Végül a 7.9. ábra az adatok a Tabulator [9] Firefox kiterjesztés által történő megjelenítését szemlélteti.

7.6. Felhasználási lehetőségek

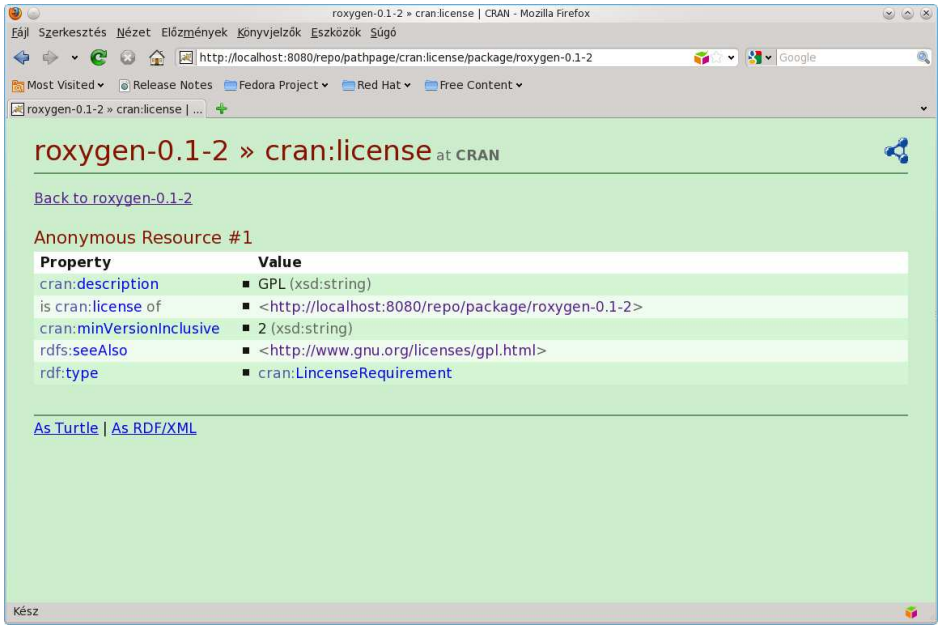
Egy csomagkezelő rendszerekben felmerülő tipikus feladat például annak megállapítása, hogy adott csomag telepítéséhez rendelkezésre áll-e minden előfeltételként szükséges további csomag. Egy szorosan kapcsolódó feladat a megfelelő sorrend meghatározása, amelyben a csomagok telepítését el kell végezni. Kézenfekvő módon merül fel a kérdés, hogy milyen segítséget adnak a webontológiák hasonló gyakorlati feladatok megoldásához. Sajnos csak minimális következtetési lehetőséget kínálnak.

A 7.4.1. szakaszban soroltuk fel a modellezési feladat kihívásait, gyakorlatilag ugyanezek a problémák nehezítik a következtetést:

- A csomagok közötti kapcsolatokban tipikusan megengedett a verzió-számok korlátozása, amely nehézkesen kezelhetővé teszi a kapcsolatokat

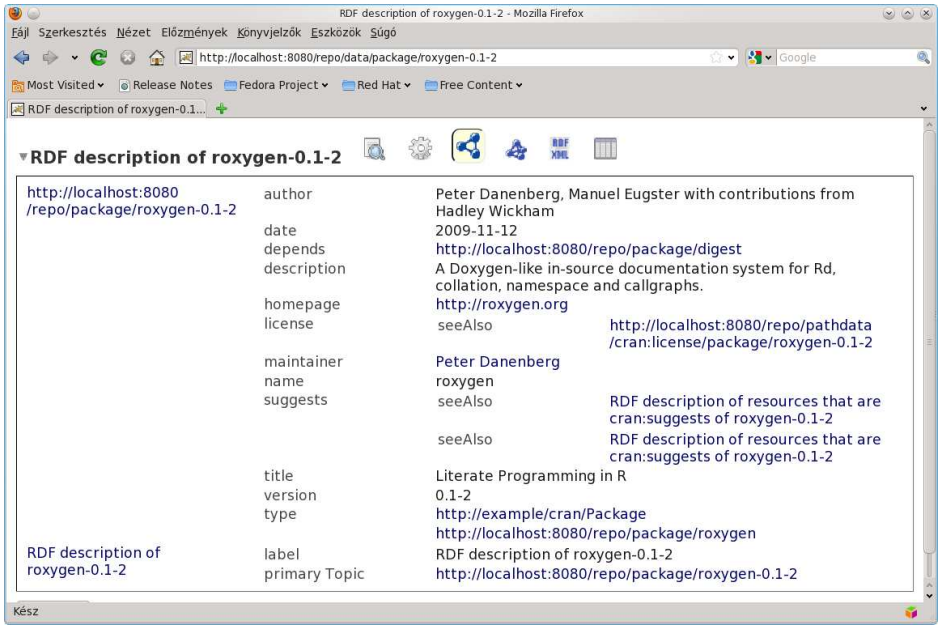


7.6. ábra. Csomag metaadatok böngészése (ajánlott csomagok)



7.7. ábra. Csomag metaadatok böngészése (licenc)

7.8. ábra. Csomag metaadatok böngészése (karbantartó)



7.9. ábra. Csomag metaadatok böngészése

az OWL és a SPARQL szempontjából.

- Verziószámok összehasonlítása speciális algoritmusokat igényel, ezek messze meghaladják az OWL és SPARQL lehetőségeit (lásd az 5.2.5. szakaszban leírtakat).

- A csomagkezelő rendszerek olyan bonyolult módon értelmezhetik a kapcsolatokat, amely rendkívül megnehezíti a megfelelő modellezést OWL-ben.[6]

Csomagok kapcsolatainak vizsgálata speciális eszközöket igényel, az RDF használata leginkább azért előnyös, mert egységes modellt biztosít szoftvercsomag metaadatok ábrázolásához.

[6]Ilyen például a Debian rendszer, amelyben bizonyos kapcsolatok speciális előfordulásainak a hagyományostól eltérő jelentése lehet. Bizonyos kapcsolatokat pedig együtt kell értelmezni, ilyenkor az együttes jelentés más, mint a külön-külön tekintett jelentések. A Debian lehetővé teszi a kapcsolatokban alternatívák megadását is, amelyek ábrázolása problémás. Alternatívák jelzéséhez a szerző jobb híján az Alt RDF konténert használja, amelyhez nem definiált formális szemantika.

Irodalomjegyzék

[1] Commons Compress. URL http://commons.apache.org/compress/.

[2] GNUpdate. URL http://gnupdate.sourceforge.net/.

[3] Joseki – A SPARQL Server for Jena. URL http://www.joseki.org/.

[4] Pubby linked data frontend. URL http://www4.wiwiss.fu-berlin.de/pubby/.

[5] RDFizers. URL http://simile.mit.edu/wiki/RDFizers.

[6] Restlet. URL http://www.restlet.org/.

[7] rpmfind.net. URL http://www.rpmfind.net/.

[8] Software Package Data Exchange (SPDX). URL http://spdx.org/.

[9] Tabulator: Generic data browser. URL http://www.w3.org/2005/ajar/tab.

[10] XPInstall. URL https://developer.mozilla.org/en/XPInstall.

[11] Dan Brickley and Libby Miller. FOAF Vocabulary Specification, 2010. URL http://xmlns.com/foaf/spec/. version 0.97.

[12] Natasha Noy and Alan Rector. Defining N-ary Relations on the Semantic Web. W3C Working Group Note, 2006. URL http://www.w3.org/TR/swbp-n-aryRelations/.

[13] Leonard Richardson and Sam Ruby. *RESTful Web Services*. O'Reilly Media, 2007. ISBN 978-0-596-80168-7.

[14] R Development Core Team. *R Installation and Administration*, 2010. URL http://cran.r-project.org/doc/manuals/R-admin.html. version 2.11.1.

[15] R Development Core Team. *Writing R Extensions*, 2010. URL `http://cran.r-project.org/doc/manuals/R-exts.html`. version 2.11.1.

[16] The Debian Policy Mailing List. *Debian Policy Manual*, 2010. URL `http://www.debian.org/doc/debian-policy/`. version 3.8.4.0.

[17] Daniel Veillard. Linux Packages Metadata Mirroring Proposal. URL `http://www.rpmfind.net/linux/rpm2html/mirroring.html`.

A. Függelék

RPM

A.1. Bevezetés

Az RPM egy rekurzív betűszó, amelynek feloldása RPM Package Manager. A név egy parancssorból használható és számos Linux disztribúció alapját képező szabad és nyílt forrású csomagkezelő rendszert takar. A betűszó egyben a rendszer által kezelt csomagok formátumát is jelenti, amely a Linux Standard Base (LSB) [1] része[1].

A rendszer eredetileg a Red Hat Linux számára készült még Red Hat Package Manager (RPM) néven [2]. Később RPM Package Manager névre keresztelték át, miután más Linux-disztribúciók is átvették használatát. 2007 óta sajnos egy „pártszakadás" nehezíti az RPM-mel kapcsolatos tisztánlátást. A `http://rpm.org/` címen elérhető hely jelenleg a 4.*x* verziósorozat[2] fejlesztésének ad otthont, míg a `http://rpm5.org/` címen egy RPM 5 nevű rendszer fejlesztése történik. Ráadásul mindkét fejlesztőközösség a saját verzióját nevezi hivatalosnak.

A két RPM projekt között alapvető különbség, hogy míg az RPM 5 projekt más platformok felé is kitárja a kapukat, addig a másik alapvetően Linux rendszereket céloz meg. Napjainkban az RPM 5 rendszer használata még kevésbé elterjedt a Linux-disztribúciók körében. Az alábbi lista a teljesség igénye nélkül vesz számba néhány RPM-alapú disztribúciót:

- RPM.org-alapú disztribúciók:
  - CentOS `http://centos.org/`
  - Fedora `http://fedoraproject.org/`

[1] A Linux Standard Base (LSB) projekt célja olyan szabványok lefektetése, amelyek kompatibilitást biztosítanak a különböző Linux disztribúciók között.

[2] A 4.*x* verziósorozat legutóbbi kiadása a stabil 4.8.1 verzió és a 4.9.0 fejlesztői verzió.

  - openSUSE `http://www.opensuse.org/`
  - Red Hat Enterprise Linux `http://www.redhat.com/rhel/`
  - SUSE Linux Enterprise `http://www.novell.com/linux/`

- RPM 5-alapú disztribúciók:
  - CAOS Linux `http://www.caoslinux.org/`
  - OpenPKG `http://www.openpkg.org/`,
  - Unity Linux `http://unity-linux.org/`

A.2. RPM csomagok felépítése

Az RPM.org és RPM 5 projektekben használt csomagok azonos felépítésűek. Minden RPM csomag egy olyan bináris állomány, amely az alábbi négy részből áll:

Állomány fejléc (lead) Az állomány első 96 bájtnyi része, amelyet ma már csak az állomány formátumának felismeréséhez használnak. Az első 4 bájt a formátumot azonosító mágikus szám, az összes többi kódolt információ rendelkezésre áll a fejléc részben.

Szignatúra (signature) Az állomány sértetlenségének és hitelességének ellenőrzéséhez használható információkat – ellenőrző összegeket és digitális aláírásokat – tartalmaz.

Fejléc (header) A csomag metaadatokat tartalmazza.

Archívum (archive, payload) A csomag állományait tartalmazó tömörített archívum.

A szignatúra és a fejléc azonos felépítésű, mindkettő azonos módon tárol metaadatokat. Történeti okokból, de nem túl szerencsés módon **fejléc struktúrá**nak (**header structure**) nevezik a szerkezetet. Ez egy olyan adatszerkezet, amely metaadat elemek tárolására szolgál az állományban. Hatékonyan kereshető meg benne bármely metaadat elemhez tartozó adat, nem korlátozza az adatok tárolási hosszát, ráadásul többféle adattípust is támogat.

A formátum a szignatúrához és fejléchez számos **címké**nek (**tag**) nevezett metaadat elemet definiál, amelyek mindegyikéhez egy előjel nélküli egész számot rendel azonosítóként. Adott szám különböző címkéket jelölhet a szignatúrában és a fejlécben. Minden címkének meghatározott továbbá a típusa,

Név	Szám	Méret (byte)
NULL	0	0
CHAR	1	1
INT8	2	1
INT16	3	2
INT32	4	4
INT64	5	nem támogatott típus
STRING	6	változó
BIN	7	1
STRING_ARRAY	8	változó
I18NSTRING	9	változó

A.1. ábra. A fejléc struktúrában rendelkezésre álló típusok

és hogy kötelező vagy opcionális a használata. Az A.1. ábra tartalmazza a rendelkezésre álló típusokat.

A fejléc struktúra három további részre bontható:

Fejléc struktúra fejléc Benne egy mágikus szám jelzi a fejléc struktúra elejét, tartalmazza továbbá az indexrekordok számát és a rekordokhoz tartozó adatok blokkjának méretét.

Index Indexrekordok alkotják. Minden rekord négy darab 32-bites egész számot tartalmaz, amelyek sorban az alábbiakat jelentik:

1. címke azonosítója
2. típus azonosítója
3. eltolás
4. elemszám

Indexrekord adatok Összefüggő blokk, amely az indexrekordokhoz tartozó adatokat tárolja.

Az indexrekordokat követő blokkból lehet kiolvasni az egyes rekordokhoz tartozó adatokat. Az indexrekordban az eltolás adja meg, hogy az indexrekord adatokat tartalmazó blokkban hol kezdődnek az adott rekordhoz tartozó adatok, az elemszám pedig azt jelenti, hogy ettől a pozíciótól hány darab megfelelő típusú adatelemet kell tekinteni.

Egy adott rekordhoz tartozó adatokat tehát úgy kaphatjuk meg, hogy az eltolás által kijelölt pozícióról n bájtot olvasunk, ahol n az elemszám és a típus tárolási méretének szorzata. Kivételt a STRING, STRING_ARRAY és I18NSTRING típusok képeznek, amelyek kezelése eltérő módon történik. A STRING típus esetében az adatok végét egy 0 értékű bájt jelzi, hasonlóan a C programozási nyelv karakterláncaihoz. A STRING_ARRAY és I18NSTRING típusok esetén pedig az elemszám által meghatározott számú olyan bájtsorozatot kell beolvasni, amelyek végét 0 értékű bájt jelzi.

Irodalomjegyzék

[1] Linux Standard Base (LSB). URL http://www.linuxbase.org/.

[2] Eric Foster-Johnson, Stuart Ellis, and Ben Cotton. RPM Guide, 2010. URL http://docs.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/.

B. Függelék

Erőforrások azonosítása

B.1. Erőforrás fogalma

A web működésének egyik legalapvetőbb fogalma az **erőforrás**, amely egy tetszőleges azonosítható dolgot jelent. A világháló felépítését és működését összefoglaló [3] dokumentum **információ erőforrások**nak (**information resource**) nevezi az információtartalommal bíró és bájtsorozatokként megtestesíthető dolgokat, amelyek hálózaton keresztül továbbíthatósága értelemszerű. Ilyenek az elektronikus dokumentumok és bináris állományok, amelyek használata mindennapos a weben. Erőforrásként lehet tekinteni azonban akár a fizikai világ objektumait és fogalmakat is, amelyek nyilvánvalóan nem információ erőforrások.

B.2. Egységes erőforrás-azonosítók

Az RFC 3896 [1] szabványban definiált URI-k erőforrások azonosítására szolgáló karaktersorozatok.[1] Az URI egy olyan általános fogalom, amelynek sokféle megjelenési formája lehetséges. A szabvány egy olyan általános szintaxist határoz meg, amelyben minden URI egy **séma-név**ből és egy **séma-specifikus rész**ből áll. A séma-specifikus rész kezelését úgynevezett **URI sémák** határozzák meg, amelyek további megszorításokat is tehetnek a séma-specifikus rész formájára. Sok felhasználás teszi lehetővé URI-k rövidítését **relatív hivatkozások**nak nevezett megfelelő URI végszeletekkel, amelyekből egy **bázis-URI** segítségével érvényes URI képezhető. Összefoglaló néven **URI hivatkozások**nak nevezik az URI-kat és relatív hivatkozásokat.

[1] Az URI betűszó feloldása **Uniform Resource Identifier**, amely szó szerint **egységes erőforrás-azonosító**t jelent.

B.3. URI-k használata

A **hivatkozás-feloldás** (**URI dereferencing**) az URI használatát jelenti az azonosított erőforrás eléréséhez. Az „elérés" kifejezés itt általános értelemben használt, valamilyen művelet végrehajtását jelenti az erőforráson. A művelet a legtöbb esetben az erőforrás információtartalmának kinyerésére irányul, de lehet akár módosítás vagy az erőforrás jellemzőinek lekérdezése. A hivatkozás-feloldás minden esetben megfelelő protokollok szerinti interakciót jelent.

Fontos hangsúlyozni, hogy egy URI betöltheti olyan formális azonosító szerepét, amely nem használható az azonosított erőforrás eléréséhez. Sok URI séma esetében eleve nem értelmezett hivatkozás-feloldás. Ha egy URI-nál lehetséges hivatkozás-feloldás, akkor sem feltétlenül történik hozzáférés a használata során.

Ha az URI hivatkozás-feloldás célja az erőforrás információtartalmának kinyerése, akkor a sikeres végrehajtás eredménye az erőforrás egy **reprezentáció**ja, amely az erőforrás aktuális állapotát szolgáltató információkat jelenti. A reprezentáció időben változhat, ráadásul egyidejűleg akár több különböző reprezentáció tartozhat egy erőforráshoz.

Tartalom-egyeztetésnek (**content negotiation**) nevezik azt a megoldást, amely lehetővé teszi egy erőforráshoz több különböző reprezentáció biztosítását, és amelyet támogat például a HTTP/1.1 [2]. Tartalom-egyeztetés segítségével a hivatkozás-feloldás során a kliens számára legmegfelelőbb reprezentáció választható. Például egy URI azonosíthat egy több különböző formátumban vagy nyelven rendelkezésre álló dokumentumot, amelynek elérésekor a kliens által előnyben részesített változat szolgáltatható reprezentációként.

Irodalomjegyzék

[1] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard), 2005. URL `http://www.ietf.org/rfc/rfc3986.txt`.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Standard), 1999. URL `http://www.ietf.org/rfc/rfc2616.txt`.

[3] Ian Jacobs and Norman Walsh. Architecture of the World Wide Web, Volume One. W3C Recommendation, 2004. URL `http://www.w3.org/TR/webarch/`.