OOo dokumentumok kezelése a Ruby segítségével

Miért kellene várnunk egy olyan operációs rendszerre, amely lehetővé teszi, hogy irodai dokumentumainkat az általunk fejlesztett alkalmazásokban használjuk fel? Dolgozzunk OpenOffice.org XML-alapú dokumentumainkkal a Ruby segítségével.

Nagyot fejlődött az utóbbi időben a szövegfeldolgozásra, táblázatkezelésre és bemutatók készítésére használható ***OpenOffice.org*** (OOo) nevű sokoldalú irodai alkalmazáscsomag. Az OOo forráskódja és fájlformátumai nevéhez méltóan teljesen nyilvánosak. Ami hatalmas előny azok számára, akik dokumentumaikat a létrehozó alkalmazások nélkül szeretnék módosítani.

OOo Extract

Először akkor figyeltem fel az OpenOffice.org dokumentumokban rejlő lehetőségekre, amikor ***Daniel Carrera*** bemutatta az ***OOoExtract*** nevű programját. Ez egy Ruby alkalmazás, melynek segítségével a parancssorból kereshetünk szövegrészleteket ***OOoWriter*** dokumentumokban. A honlapja szerint az OooExtract segítségével a szövegben és a stílusok között is kereshetünk, akár helyettesítő karakterekkel is, emellett a program képes logikai műveletekből felépített keresések futtatására is. A program minden olyan operációs rendszeren fut, melyhez létezik Ruby fordító.
A Rubyról már esett szó a Linuxvilágban. Aki nem ismeri, annak legyen elég annyi, hogy leginkább a Perl és a Smalltalk közti átmenetről van szó, némi Lisp és Python beütéssel. Erősen objektum-alapú, formanyelve tiszta és intuitív. Szerzője, ***Yukirhiro „Matz" Matsumoto***, az első alfaváltozatot 1994-ben adta ki.
Az OOoExtract kipróbálásához töltsük le a programot. Jelenleg az alkalmazást egyetlen futtatható fájlként, vagy tarcsomagként tölthetjük le, melyben a szükséges könyvtárak külön fájlokban találhatók. Telepítés után létrehozhatunk egy ***Writer*** dokumentumot és abban kereséseket hajhatunk végre. Ha kéznél van az OOo, akkor indítsuk el és írjunk be valami rövid szöveget, például:

```
Ez egy példa
Két sorból áll
```

Mentsük a fájlt ***pelda1.sxw*** néven ugyanabba a könyvtárba, ahová az OOoExtractot telepítettük és indítsuk el az OOoExtractot a parancssorból:

```
./ooo_extract.rb –text példa pelda1.sxw
Ez egy példa
```

A program ekkor a ***pelda1.sxw*** fájlban keresi a „példa" szót. Ez valójában egy egyszerű kifejezés. Bonyolultabbakat is használhatunk, például az alábbit, mely a három karakterből álló szavakat keresi:

```
./ooo_extract.rb –text "\s\w\w\w\s"  sample1.sxw
Két sorból áll
```

Ez így nagyszerű, de az OOoExtract egyik legnagyszerűbb tulajdonsága mégis az, hogy a metaadatok, azaz a dokumentumban lévő szöveggel kapcsolatos adatok között is kereshetünk. Tegyük fel, hogy a példaként felhozott Writer dokumentumot az alábbi sorral egészítjük ki:
Itt már a formázáson is változtattunk
A szöveg begépelése után jelöljük ki a „már" szót, stílusát állítsuk át a Footer (lapalji jegyzet) stílusra és futtassuk le az alábbi keresést:

```
./ooo_extract.rb –style="Footer"  pelda1.sxw
Itt már a formázáson is változtattunk
```

Az OOoExtracttel tehát nemcsak a szöveg tartalmában kereshetünk, de a megadott formázási stílushoz tartozó szövegrészleteket is előbányászhatjuk. Így az OOoExtract akár a tartalomra és a jelentésre épülő kereséseket is végezhet, ezáltal a Writer dokumentumból egyszerű adatbázist hozva létre.
A programot egyszerre több fájlon is lefuttathatjuk, ha a fájlnévben helyettesítő karaktereket használunk. Tegyük fel, hogy Writer fájlokban recepteket tárolunk. Ha a szövegeket egyéni stílusokkal készítettük el, akkor például megkereshetjük, hogy melyik receptben szerepel az alma mint hozzávaló:

```
./ooo_extract.rb –text="alma" –style="Hozzávaló"
recipes/*.sxw
AlmasSalsa.sxw: 2 közepes piros alma
AlmasRetes.sxw: 4 csésze hámozott és szeletelt alma
```

Az SXW fájlformátum

Hogyan képes az OOoExtract erre a varázslatra? A titok a fájlformátumban rejlik. Bár minden Writer fájl kiterjesztése .sxw, a file nevű UNIX parancsot alkalmazva kiderül, hogy zip fájlról van szó:

```
$ file pelda1.sxw
pelda1.sxw: Zip archive data, at least v2.0 to
➥extract

És mi található benne? Lássuk:
$ unzip -l pelda1.sxw
Archive:  pelda1.sxw
  Length     Date   Time    Name
 --------  -------- -----
       30  11-26-03 01:40   mimetype
     2328  11-26-03 01:40   content.xml
     8358  11-26-03 01:40   styles.xml
     1159  11-26-03 01:40   meta.xml
     7021  11-26-03 01:40   settings.xml
      752  11-26-03 01:40   META-INF/manifest.xml
 --------                   ----
    19648                   6 files
```

Az OOo XML formátumában minden tartalom és metaadat egyszerű szövegként kerül tárolásra, nem kell aggódnunk a titokzatos bináris kódolás vagy az átláthatatlan szerkezetek miatt. Mivel az adatok XML formában vannak tárolva, az OOo fájljait sok más programmal is beolvashatjuk. Az egyszerű szövegként való tárolás természetesen azt is jelenti, hogy a fájllal kapcsolatos adatok magában a fájlban tárolódnak, egyszerűen olvasható formában. Természetesen az OpenOffice.org munkatársai nem hagynak minket magunkra, mert a dokumentációban a fájlformátum részletes leírása is megtalálható. A OpenOffice.org XML 1.0-és felhasználói kézikönyve egy 571 oldalas PDF fájl. Bevallom, hogy nem olvastam el az egészet, de kétlem, hogy a legapróbb részlet is hiányozna belőle.
Példánk kedvéért csupán néhány alapvető szabályt tekintünk át, melyek segítségével felfedhetjük az OOoExtract működését, illetve a fájlformátumot is megismerhetjük.
A példadokumentum kicsomagolása után a ***content.xml*** fájlt töltsük be egy szövegszerkesztőbe. Ekkor észrevehetjük, hogy a fájl nem túl áttekinthetően van formázva. Futtassunk a fájlon egy XML formázó segédprogramot (például tidy-t), amely sorkihagyásokat és behúzásokat helyez el a szövegben.
A fájl egy XML meghatározással kezdődik, melyet egy DOCTYPE hivatkozás követ, közvetlenül utána pedig a gyökérelem áll, az `office:document-content`. A kezdő tag számos XML névtér tulajdonságot tartalmaz, melyekkel nem kell foglalkoznunk, de áttekintve őket már lehet némi elképzelésünk az OOo dokumentumokban található tartalom felépítéséről.
Közvetlenül a gyökérelemben a szkriptek, betűtípus-meghatározások és stílusok gyerekelemeit találjuk. Mivel példánk egyszerű dokumentum, itt nem sok adatot találunk. Számunkra most az `office:body` elem tartalma fontos. Még ebben a fontos fájlban is viszonylag kevés elem fejezi ki bizonyos alkotóelemek, például táblázatok és ábrák hiányát vagy meglétét. A teljes dokumentum hozzáférhető a CD melléklet Magazin/OOo könyvtárban.
A dokumentum tényleges tartalma text:p elemekben található:

```
<text:p text:style-name="Standard">Ez egy
➥példa</text:p>
<text:p text:style-name="Standard">Két sorból
➥áll</text:p>
<text:p text:style-name="Footer"> Itt már
➥a formázáson is változtattunk</text:p>
```

Ha véletlenül nem ismernénk az XML formanyelvét, a fenti sorok a text (szöveg) névtérben meghatározott p (bekezdés) elemek. Az előtag és a kettőspont segítségével hivatkozunk a dokumentum tetején meghatározott névtér URI-ra, így elkerülhetjük a más XML szóhasználatokhoz meghatározott p elemekkel való ütközést. Példánkban ezt egy teljes elemnévként tekintjük.
A példa csak három bekezdést tartalmazott, tehát három p elemre számíthatunk. Mindegyik rendelkezik egy text:stílusnév tulajdonsággal, mely a bekezdés szövegére vontakozó stílust jelöli meg. Az OOoExtract ezen tulajdonság segítségével találja meg a keresett stílussal írt szövegrészleteket.
Vajon mi lehet a Footer (lapalji jegyzet) stílusban?
A stílusmeghatározást nem a content.xml fájl tartalmazza, s ez az elkülönítés jó dolog. Mennyivel kényelmetlenebb lenne, ha egy egyszerű név helyett minden esetben a stílus teljes leírása (betűméret, szín stb.) állna itt. Így elvesztenénk a tartalom alapján történő keresés lehetőségét, s az adatokkal csak a leképezés szintjén foglalkozhatnánk. Aki kíváncsi a Footer meghatározására, az kukkantson bele a styles.xml fájlba. Ebből kiderül, hogy a Footer a Standard stílusból épül fel, az alapstílus némi módosításával.

Zipből REXML

Valóban nagyszerű, hogy az OpenOffice.org tömörített XML fájlokat használ, de mi következik azután, hogy a csomagot kibontottuk? Szerencsére a Ruby 1.8 tartalmaz egy remek XML olvasót. A REXML egy, az XML 1.0-nak megfelelő olvasó, mely a Rubyra emlékeztető API-ja mellett az XPath és a SAX2 teljes megvalósítását is magában foglalja. Fejlesztője ***Sean Chittenden***, aki bevallása szerint azért írta, mert akkoriban csak a Rubyban csupán két módszer létezett az XML fájlok olvasására. Az egyik a natív C olvasóhoz való kapcsolódás, melynek hátránya a hordozhatóság elvesztése. A másik, hogy tisztán csak a Rubyt használjuk, Sean szerint azonban a Ruby ehhez nem rendelkezett megfelelő API-val. Sean jól ismerte a különféle Java-alapú XML olvasókat, azonban ezek a W3C-féle DOM-hoz vagy a közösségi SAX-hoz igazodnak, s ez nem tesztett neki. Az Electric XML feljesztői egy olyan, a Java szókincsére épülő API-t tettek elérhetővé, melyhez a Java programozók hamar hozzászokhatnak.
Ugyanez az elv állt az REXML (Ruby Electric XML) API fejlesztése mögött is . Nem meglepő módon a REXML API az eredeti Java-stílustól fokozatosan eljutott a Ruby-szerű felépítéshez, így a fejlesztők a Rubyból ismert formanyelv és programozási fogások (például a blokkok és beépített eszközök) használatával képesek XML fájlokat kezelni.

A REXML API

A REXML faolvasó segítségével egyszerűen betölthetjük az XML dokumentumokat:

```
require "rexml/document"
file = File.new( "xml_fájl.xml" )
```

```
doc = REXML::Document.new file
vagy:
require "rexml/document"
my_xml_string = "<sample>
   <text>Ez a mi REXML dokumentumunk</text>
   </sample>"
doc = REXML::Document.new my_xml_string
```

A Document konstruktor karakterláncot vagy I/O objektumot fogad. A REXML felismeri, hogy melyik esetről van szó és eszerint cselekszik. Ha megvan a kérdéses dokumentum, az elemek megtalálásához a Ruby tömbjeit és az `each` kulcsszót XPath választóval kell használnunk:

```
my_xpath = "sample/text"
doc.elements.each( my_xpath ){
    |el| puts el.text }
```

A fenti példában az each metódus az XPath választóra illeszkedő összes elemen végighalad. Minden egyes meghívásnál egy kódblokk (a kapcsos zárójelek közötti rész) kerül meghívásra. Az el változó az éppen vizsgált elemet jelenti, tehát a példa az XPath-ra illeszkedő elemekhez tartozó szöveget jeleníti meg.

XPath

Writer példaszövegünk és a hozzá tartozó XML fájl meglehetősen egyszerű, így a megfelelő stílushoz tartozó elemek megtalálása sem túl bonyolult feladat. Egy ilyen egyszerű példa leginkább a jelen cikkhez hasonló szövegekhez használható, de a valóságban kevéssé valószínű, hogy ilyennel találkozunk: lehet, hogy a leíró rendszert csupán részleteiben ismerjük. Az ilyen fájlokban való keresés már nagyobb kihívást jelent, de az XPath minden gondunkat megoldja. Az XPath a W3C javaslata az XML dokumentumok részeinek eléréséhez. Segítségével egy útvonalmeghatározást hozhatunk létre, mely a helymeghatározást az elemek, tulajdonságok neve, tartalma, illetve relatív vagy abszolút helyzete alapján végzi el. Összetett XML dokumentum esetén például az alábbi sorral egy XPath kifejezést hozhatunk létre, mely az összes olyan `text:p` elemet megtalálja, mely az `office:body` elem közvetlen leszármazottja:

```
*/office:body/text:p
```

XPath nyelven a kezdő * karakter azt jelenti, hogy kövessünk az XML dokumentumban minden olyan `text:p` elemekhez vezető útvonalat, ahol a `text:p` elem az `office:body` elem gyermeke. REXML-ben pedig az alábbi XPath segítségével válogathatunk az illeszkedő elemek között:

```
xml.each_element( */office:body/text:p" ) do |el|
   # Az el változóval csinálunk valamit, például
   # valamilyen tartalom vagy stílustulajdonság
után kutatunk
end
```

A fenti példában a do és az end között egy blokk szerepel. Névtelen függvényre hasonlít, mely a csoport minden egyes eleménél (jelen esetben az XPath-ra illeszkedő elemeknél) meghívásra kerül. Az elem pedig paraméterként kerül átadásra, ezt jelzi a do után álló két függőleges vonal is. Lényegében így működik az OOoExtract, de a számos parancssori kapcsoló működéséről még többet olvashatunk a program honlapján.

A cél egy általános OOo API

Az OOoExtract megismerése után rájöttem, hogy a Rubyhoz egy általánosabb célú OOo objektumra van szükségem. Az OOoExtract mögött álló alapelvek nem csak az adatok olvasását teszik lehetővé, hanem segítségükkel például az adatbáziseszközökből ismert és kedvelt CRUD műveletek létrehozása, frissítése és törlése is megvalósítható. E célból jött létre az OOo4R fejlesztés a RubyForge-on, a Ruby szoftveres CVS gyűjteményében. A cél az adatok és metaadatok egyszerű elérése, az XPath rugalmas, észrevétlen használata és egy API létrehozása a közhelynek számító műveletek elvégzésére. Helyhiány most csak a metaadatok elérésével foglalkozunk részletesebben, hiszen itt tulajdonképpen arról van szó, hogy a Ruby dinamikus üzenetkezelésének segítségével keresünk az elemek tartalmában.
Korábban láthattuk, hogy egy OOo dokumentum több XML fájlt tartalmaz, melyek egy zip fájlban kerülnek tárolásra. A ***content.xml*** fájllal már foglalkoztunk, egy másik pedig a ***meta.xml*** névre hallgat. Ez utóbbi magával a dokumentummal kapcsolatos adatokat tartalmaz, például a címét, a létrehozás időpontját és a szavak számát. A gyökérelem az ***office:document-meta***, ezen belül helyezkedik el az `office:meta` elem, mely az értékes adatokat tartalmazó származékelemek szülője:

```
<meta:initial-creator>James Britt
</meta:initial-creator>
<meta:creation-date>2003-11-25T17:36:31
</meta:creation-date>
<dc:creator>James Britt</dc:creator>
<dc:date>2003-11-25T18:40:59</dc:date>
<dc:language>en-US</dc:language>
<meta:editing-cycles>13</meta:editing-cycles>
```

A teljes metaadatfájl megtalálható a CD melléklet Magazin/OOo könyvtárában.
A fő Document osztály mellett az OOo4R a metaadatokat magában foglaló meta osztályt is meghatároz. Egy meta osztály egy REXML dokumentum használatával tárolja a ***meta.xml*** fájl tartalmát. Egy meta objektum jórészt tulajdonságok gyűjteménye. Általában arra van szükségünk, hogy lekérdezünk bizonyos adatokat, például a szerző nevét, vagy beállítunk egy új értéket, például új címet adunk meg. Minden tulajdonsághoz két metódus szükséges, de használhatunk dinamikus metódusmeghívást is. Ez utóbbi azt jelenti, hogy elfogjuk az elérési kérelmeket, találunk rájuk illeszkedő tulajdonságneveket és végrehajtjuk a kért műveletet vagy megszakítást kezdeményezünk.
Az alábbi kódpélda az OOo-ban használt Dublin Core metaadat-elemekkel foglalkozik. A Dublin Core Metaadat Kezdeményezés (Dublin Core Metadata Initiative) a metaadatokkal kapcsolatos szabványalkotás nyílt fóruma. Dublin Core elemeket gyakran találunk RSS adatokban és néhány XHTML dokumentumban is előfordulnak. Az

OpenOffice.org XML fájljaiban lévő elemekhez hasonlóan ezek is rendelkeznek névtér-előtaggal. Ezeket az előtagokat nem kell megjegyeznünk, a teljes elemnevet hozzárendelhetjük valami egyszerűbben megjegyezhető névhez.
A Meta osztály meghatározása egy hash létrehozásával kezdődik, mely a valódi elemnevekhez barátságosabb neveket, illetve a metaadatok alap XPath-jét tartalmazó osztálykonstanst rendel. Az osztály alkotófüggvénye (konstruktora) egyszerűen REXML dokumentumot készít az XML forrásból:

```
module OOo
  class Meta

  NAME_MAP = {
   ´description´ => ´dc:description´,
   ´subject´     => ´dc:subject´,
   ´creator´     => ´dc:creator´,
   ´author ´     => ´dc:creator´,
   ´date´        => ´dc:date´,
   ´language´    => ´dc:language´,
   ´title´       => ´dc:title´
  }
    XPATH_BASE  = “*/office:meta”

    def initialize( src )
      @doc = REXML::Document.new(  src.to_s )
    end
```

A `method_missing` nevű metódust újra meghatározhatjuk úgy, hogy minden Ruby osztály számára elérhető legyen. Ilyenkor nem megszakítás jön létre, hanem a metódus megnézi, hogy az objektumhoz küldött üzenet illeszkedik-e a metadaat valamely elemére:

```
def method_missing( name, *args )
  n = name.to_s
  if is_assignment? n
    el = map_for_assignment n
    xpath = ”#{XPATH_BASE}/#{el}”
    assign( xpath, *args)
  else
    el = Meta.map_name n
    xpath = ”#{XPATH_BASE}/#{el}”
    find( xpath  )
  end
end
```

A `method_missing` első paramétere egy szimbólumobjektum, tehát a kód a helyettesítő karakterláncot olvassa be. Az `is_assignment` metódus ellenőrzi, hogy a név egyenlőségjelre végződik-e. Ha igen, akkor hozzárendelésról van szó és ekkor a `map_for_assigment` a metaadat neve után álló minden jelet töröl, a barátságosabb nevet pedig a Dublin Core elemnévhez kapcsolja. Az `assign` a REXML dokumentumban frissíti a megfelelő elemet:

```
def assign( xpath, val )
  node = @doc.elements.to_a( xpath )[0]
  node.text = val
end
```

Ha nem hozzárendelésről van szó, a kód megpróbálkozik a metaadatok beolvasásával. Akárcsak előzőleg, a név itt hozzárendelésre kerül, de most a kód a `find`-ot hívja meg:

```
def find( xpath )
 begin
  return @doc.elements.to_a( xpath.to_s )[0].text
 rescue Exception
  raise OOo::OOoException.new(
      “Error with xpath ´#{xpath}´: #{$!}”, $@ )
 end
end

# Kihagyott segítőmetódusok...

 end
end
```

E módszerrel a többi metaadat-elemet is elérhetjük, bár van néhány különleges eset, ahol a metaadatokat egy csapat származékelem tartalmazza. A módosított OOo dokumentumok mentéséhez a Ruby beépített Zip osztályának segítségével frissítsük a zip fájl tartalmát és írjuk vissza a lemezre.

Összefoglalás

Mivel az OpenOffice.org fájlformátum egy teljesen dokumentált XML formátumot használ, OOo fájlokat az OOo nélkül is létrehozhatunk és módosíthatunk. A Ruby beépített dinamikus XML kezelése tökéletesen alkalmassá teszi a programot az OOo dokumentumok kezelésére.

Linux Journal 2004. március, 119. szám

James Britt az arizonai Scottsdale-ben működő, szoftverfejlesztéssel és tervezéssel foglalkozó vállalat, a Neurogami vezetője. Részt vett egy XML-ről szóló, a Wrox Press által kiadott könyv megírásában, emellett számtalan szoftverfejlesztéssel foglalkozó cikk származik tőle, valamint a Rubyról és XML-ről előadást tartott a texasi Austinban rendezett 3. Nemzetközi Ruby Konferencián.
A jamesgb@neurogami.com címen érhető el.

FORRÁSOK

Dublin Core: dublincore.org
OOo_extract:
www.math.umd.edu/~dcarrera/openoffice/tools/ooo_extract.html
OOo formátumok:
xml.coverpages.org/starOfficeXML.html
OpenOffice.org XML: xml.openoffice.org
Ruby:
linux.oreillynet.com/pub/a/linux/2001/11/29/ruby.html
RubyForge: www.rubyforge.org
“Gondolkodj XML-ben: az Open Office fájlformátuma”:
www-106.ibm.com/developerworks/xml/library/x-think15
XPath: www.w3.org/TR/xpath