Debreceni Egyetem
Informatikai Kar

APACHE MAVEN BŐVÍTMÉNY FEJLESZTÉS

Témavezető:
Jeszenszky Péter
egyetemi adjunktus

Készítette:
Zsolczai Gergő
Programtervező informatikus

Debrecen
2011

Tartalomjegyzék

1. fejezet

Bevezetés

Bevallom, mikor kiválasztottam szakdolgozatom témáját, nem sokat tudtam az *Apache Maven*ről, és a Szemantikus web koncepciójáról sem hallottam még. Viszont úgy éreztem, a Java nyelv már közel áll hozzám és abban a félévben épp a konzulensemhez jártam Programozási környezetek gyakorlatra, ahol az első órán nagy vonalakban elmagyarázta, hogy mit is tud a *Maven*. Így megláttam benne a lehetőséget, és úgy gondoltam, hogy belevágok. Később, ahogy gyakorlaton eljutottunk ehhez a témához, minél többet tudtam meg róla, annál jobban megtetszett, annál inkább beláttam, milyen nagymértékben segíti egy Java szoftverprojekt életciklusának implementációs és tesztelési fázisát. Ahogy a félév végéhez közeledtem, egyre inkább úgy tűnt, jó témát választottam szakdolgozatomhoz, főleg mikor megtudtam, az *Apache Software Foundation* weboldala [2] nagy segítséget nyújt abban, hogyan is kell implementálni egy bővítményt *Maven* rendszeréhez. Ezen oldal áttanulmányozása után biztosra vettem, nem is lesz olyan nehéz dolgom.

Már csak azt kellett kitalálnom, mi legyen pluginom funkcionalitása. Azzal tisztában voltam, nem kell túl bonyolultnak lennie ahhoz, hogy hasznos legyen, elég, hogyha egy kis dologgal egészítem ki a *Maven*t. Ezután keresni kezdtem, hogy a Programozási környezetek gyakorlaton megtanult pluginokon kívül milyen bővítmények léteznek még ehhez a projektkezelő rendszerhez. Mivel annyiféle plugint találtam, amik az általam ismert feladatokhoz hozzá voltak rendelve, és azokat nagyon jól megvalósították, sokáig nem is tudtam, milyen pluszt adhatnék hozzá a *Maven*-hez.

Szerencsére számíthattam konzulensem tanácsára, így mikor problémámmal felkerestem, felvetett egy ötletet: Mi lenne, ha témámat összekapcsolnám a szemantikus világhálóval? Persze fűzött hozzá magyarázatot is, de elsőre elég nagy falatnak tűnt nekem ez a téma. Elmondta, hogy ő tart Szemantikus Web kurzust, úgyhogy jegyzettel, példákkal, és egy nagyon jó könyvvel is segíteni tudja munkámat.

Következő alkalommal, mikor felkerestem, már kölcsön is tudta adni a könyvet [13], mely nagyonban hozzájárult ahhoz, hogy megértsem, mit is jelent a szemantikus web fogalma, és miért van rá szükség. Mielőtt még e fogalom ismertetésére rátérne, bevezetésképpen bemutatja hogyan néz ki manapság a világháló, elmagyarázza, hogyan működnek a hagyományos keresőrendszerek és a Google *PageRank* algoritmusa, és rávezet a web nagy hiányosságára, mégpedig, hogy az információk nagyrészt struktúrálatlan formában találhatók meg rajta. A témával kapcsolatbani tudásom elmélyítése érdekében elolvastam témavezetőm RDF-ről szóló jegyzetét is, amely weboldaláról letölthető [12]. *Edd Dumbill* három *XML Watch: Describe open source projects with XML* cikkének [7] [8] [9] áttanulmányozása után már készen álltam, hogy elkezdjem Apache Maven bővítményem, a *zsolczai-doap-maven-plugin* fejlesztését.

A bővítmény elkészülte után írtam ezt a dokumentumot, melyben összefoglalom a témával kapcsolatban megszerzett tudásomat, majd bemutatom az általam készített plugint.

2. fejezet

Szemantikus Web

A szemantikus világháló alapötlete *Tim Berners-Lee*-től származik, akitől a Word Wide Web, és a hozzá kapcsolódó technológiák is (HTTP, HTML, stb). Ezen elgondolás lényege, hogy az világháló forrásaihoz (weboldalak, képek, online megnézhető videók, stb.) metaadatokat[1] rendeljünk, pontosabban struktúráltan megjelenő metaadatokat.

Ez az elgondolás már korábban is megjelent, néhány technológiában már találkozhatunk velük, hiszen az összes fájlrendszer tartalmaz metaadatokat, például az állományok utolsó módosításának dátumát és a hozzáférésre vonatkozó jogokat. Az MP3 fájlok esetén az `id3v1` és `id3v2` elemek is metainformációk. Weboldalakban is fellelhetők ilyen információk, ezek a `META` HTML elemek. Itt az oldalhoz tartozó kulcsszavakat adhatunk meg, a keresőrobotok számára hasznos információkat, amik segítségével könnyebben kategorizálhatják weboldalunkat. Ez az elem azonban nem elég a szemantikus web megvalósításához, mivel nem írhatunk le elég részletes kapcsolatokat vele, azt nem adhatjuk meg vele, hogy egy kulcsszóként feltüntetett fogalom vagy tárgy milyen viszonyban van weboldalunkkal, csak azt, hogy szerepel benne. A szemantikus világháló célja, hogy egy oldal minden egyes objektumáról lehessen információkat közzétenni, például mik, esetleg kik szerepelnek egy oldalon lévő fényképen, ki és mikor készítette ezt a képet, stb.

A másik ötlet, hogy ezen metainformációk alapján következtetni lehessen. Például, ha egy kép metainformációi közt a kép szereplőjének *macska* van megadva, akkor azt a képet találatként akkor is jelenítse meg egy kereső, ha mi *cicát*, *emlőst*, vagy esetleg *állatot* kerestünk.

A továbbiakban bemutatom a szemantikus web megvalósításához szükséges technológiákat.

[1] adatokról szóló adat

2.1. XML

Az XML[2] célja, hogy az adatokat struktúrált formában írjuk le olyan szöveges dokumentumokban, hogy az számítógép által is feldolgozható legyen. Ez biztosítja a gépek közti adatcseréjét, mivel az XML platformfüggetlen, és mivel szöveges dokumentumokat használ, ezek ember által is olvasható, hiba esetén könnyen javíthatók. Lényegében ez egy leíró nyelv, vagyis dokumentumok leírására szolgál. HTML és az XML dokumentum kísérteties hasonlóságának oka a közös ős, az SGML[3]. A HTML valójában ennek a leíró nyelvnek az alkalmazása. Ez a nyelv a weben történő széleskörben használathoz, ezért hozták létre az XML-t az SGML egyszerűsítéséből.

Az XML egyik hibája a bináris tárolással szembeni karakteres tárolásból adódó nagy fájlméret. Emiatt sokan ellenszenvesnek találják ezt a reprezentációt, a többiek viszont nem foglalkoznak vele, hiszen a tárolókapacitás egyre inkább olyan méreteket ölt, hogy akár egy néhányszáz megabájtos XML állomány is elfogadható. Ráadásul az adatátviteli eszközök sávszélessége is elég nagy ahhoz napjainkban, hogy XML alapokon kommunikáljunk rajtuk keresztül.

2.1.1. Szintaxis

Egy XML dokumentum elemekből és attribútumokból áll, csakúgy, mint egy HTML dokumentum, viszont a HTML szabvány rögzíti, hogy milyen elemeket és attribútumokat használhatunk. Példa egy egyszerű XML dokumentumra:

```
<?xml version="1.0" encoding="UTF-8"?>

<hallgatok>
    <hallgato>
        <nev>Gipsz Jakab</nev>
        <neptun>G1PS2J</neptun>
        <kar>IK</kar>
        <szak>PTI</szak>
        <evfolyam>2<evfolyam>
        <telefonszam>06707070707</telefonszam>
    </hallgato>
    <hallgato>
        <nev>Wincs Eszter</nev>
        <neptun>W1NCS3<neptun>
        <kar>IK</kar>
        <szak>MI</szak>
```

[2] eXtensible Markup Language – Kiterjeszthető Jelölőnyelv

[3] Standard Generalized Markup Language – Szabványos Általánosított Jelölőnyelv

```
        <evfolyam>1</evfolyam>
    </hallgato>
</hallgatok>
```

Az első sor tekinthető a dokumentum fejlécének, ahol megadható a karakterkódolás. Ezen példán láthatjuk, hogyan is lehet struktúrált adatokat megadni XML-ben. A dokumentumban pontosan egy gyökérelem szerepelhet, a többi elem előfordulása attól függ, milyen megszorításokat adunk rá a *sémanyelvek* segítségével. Mivel telefonszám csak egy hallgatónál van megadva, láthatjuk, hogy ez az elem opcionális, tehát nem kötelező megadni. Ez a hierarchia egy fát határoz meg, aminek gyökere a gyökérelem.

Egy elemnek tetszőleges számú attribútuma lehet, de mindegyikből csak egy. Az, hogy XML dokumentumunkban melyik elemnek adunk attribútumot, hány attribútum van, esetleg egyáltalán nincs attribútum a dokumentumban, lényegtelen, mert attribútumok helyett is használhatunk elemeket. Ilyen szempontból egy dokumentum megtervezése csupán attól függ, hogyan lesz emberi szemmel átláthatóbb és hogyan könnyebb feldolgozni. Viszont azt vegyük figyelembe, hogy attribútum értéke csak szöveg lehet. Az előző példa attribútumokkal:

```
<?xml version="1.0" encoding="UTF-8"?>

<hallgatok>
    <hallgato neptun="G1PS2J" nem="ferfi">
        <nev>Gipsz Jakab</nev>
        <kar>IK</kar>
        <szak>PTI</szak>
        <evfolyam>2<evfolyam>
        <telefonszam>06707070707</telefonszam>
    </hallgato>
    <hallgato neptun="W1NCS3" nem="no">
        <nev>Wincs Eszter</nev>
        <kar>IK</kar>
        <szak>MI</szak>
        <evfolyam>1</evfolyam>
    </hallgato>
</hallgatok>
```

Ha egy XML dokumentumban egy elem neve egyedi, attól még nem biztos, hogy több XML dokumentum összefésülésekor az a név egyedi lesz, előfordulhatnak névütközések. Ezen névütközések elkerülésére dolgozták ki az *XML-névterek*et. Ezek a névterek előre definiált elemeket és attribútumokat tartalmaznak, amiket URI-kkal[4] jelölünk. Egy URI általában egy szervezethez vagy személyhez tartozik, és szerepelhetnek benne olyan karakterek is, amik nevekben nem

[4]Uniform Resource Idenfitier – Egységes Erőforrás Leíró

megengedettek. Mivel az URI-k általában nagyon hosszúak, és tartalmazhatnak olyan karaktereket, amelyek az XML elemekben nem megengedettek, bevezették az `xmlns` attribútumot, amivel ilyen formában a dokumentum alapértelmezett névterét lehet deklarálni. Ez azt jelenti, hogy a dokumentumban az előtag nélkül megadott elemek az alapértelmezett névtérbe tartoznak. `xmlns:`*prefix* formában pedig a *prefix* előtaghoz rendelt névteret lehet deklarálni, ahol a *prefix* szintén a nevekben megengedett karakterekből állhat. Példa névterek használatára:

```
<?xml version="1.0" encoding="UTF-8"?>

<hallgatok xmlns="http://www.fiktiv.egyetem.hu/"
        xmlns:t="http://www.telefon.hu/">
    <hallgato neptun="G1PS2J" nem="ferfi">
        <nev>Gipsz Jakab</nev>
        <kar>IK</kar>
        <szak>PTI</szak>
        <evfolyam>2<evfolyam>
        <t:mobilszam>06707070707</t:mobilszam>
    </hallgato>
    <hallgato neptun="W1NCS3" nem="no">
        <nev>Wincs Eszter</nev>
        <kar>IK</kar>
        <szak>MI</szak>
        <evfolyam>1</evfolyam>
    </hallgato>
</hallgatok>
```

2.1.2. Tartalmi ellenőrzés

Minden dokumentumnak definíció szerint *jól formázott*nak kell lennie, meg kell felelnie bizonyos jól formázottsági megszorításoknak. Sok ilyen megszorítás van, ezek közül a legfontosabbak:

- minden nyitó címkének rendelkeznie kell egy záró címke párral,
- minden elemnek megfelelően egymásba skatulyázottnak kell lennie.

Néha azonban nem elég ha egy dokumentum jól formázott, mert szerkezetre is eleget kell tennie bizonyos megszorításoknak, amelyeket szintén ellenőrizni kell. Ezeket a megszorításokat XML sémákkal adhatjuk meg.

2.1.3. XML sémák

Több XML sémanyelv is létezik, mellyel megszorításokat tehetünk a tartalomra és a szerkezetre. Manapság a két leghasználatosabb XML sémanyelv a W3C XML Schema és a RelaxNG. XML sémák segítségével megadhatjuk, hogy a hozzájuk tartozó XML dokumentumokban milyen elemek és attribútumok lehetnek, azokban milyen értékek megengedettek, az egyes elemeket kötelező-e megadni, stb. XML séma segítségével a feldolgozó alkalmazás könnyedén ellenőrizheti, hogy az általa feldolgozandó XML dokumentum formátuma helyes-e, a dokumentumban megadott adatok megfelelő típusúak-e, így a feldolgozóba nem kell bonyolult ellenőrző kódot írni elágaztatásokkal és kivételkezeléssel. Egy primitív XML sémanyelvnek tekinthető a DTD is az XML 1.0 részeként. Példa XML sémára:

```
<?xml version="1.0" encoding="UTF-8"?>

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="hallgatok">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="hallgato" type="xs:string"
                    minOccurs="1" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:attribute name="neptun"
                            type="xs:string" use="required"/>
                    <xs:attribute name="nem"
                            type="nemTipus" use="required"/>
                    <xs:sequence>
                        <xs:element name="nev"
                                type="xs:string">
                        <xs:element name="kar"
                                type="xs:string">
                        <xs:element name="szak"
                                type="xs:string">
                        <xs:element name="evfolyam"
                                type="xs:unsignedByte">
                        <xs:element name="mobilszam"
                                type="telefonszamTipus"
                                minOccurs="0"
                                maxOccurs="1">
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
```

```
</xs:element>

<xs:simpleType name="telefonszamTipus">
    <xs:restriction base="xsd:string">
        <xs:pattern value="\+?[0-9]+([- ][0-9]+)*"/>
    </xs:restriction>
</xs:simpleType>

<xs:simpleType name="nemTipus">
    <xs:restriction base="xs:string">
        <xs:enumeration value="ferfi"/>
        <xs:enumeration value="no"/>
    </xs:restriction>
</xs:simpleType>
```

Bővebben a W3C XML Schema típusairól [11] és a séma által nyújtott további lehetőségekről [10] a W3C XML Schema specifikációjában olvashatunk.

2.1.4. XSLT

Ezzel a nyelvvel XML dokumentumokat transzformálhatunk más XML dokumentumokká. Az XSLT[5] segítségével kifejezett transzformációkat stípuslapoknak (stylesheet) nevezzük. A transzformáció mintaillesztés segítségével történik. A transzformáció során elhagyhatunk elemeket, de akár újakat is létrehozhatunk. A mintákat az XPath nyelv segítségével írhatjuk le. Az XSLT és az XPath egy közös nyelvcsaládba, az XSL[6]-be tartoznak. A technológia lényege, hogy különválasszuk az adatokat és a megjelenítést. Az adatokat XML-ben, míg a megjelenítést XSLT-ben írjuk le.

2.1.5. XML összefoglalás

Az XML-t nevezhetjük a szemantikus világháló előfutárának. Alkalmazása manapság már elég széleskörű, elterjedéséhez elég volt néhány év. Alapvetően adatcsere formátum, de már a Microsoft Office is ebben az alakban menti dokumentumait. A web elterjedése után jött létre, és még így is nagymértékben megváltoztatta a számítógépek közti kommunikációt, mivel ezelőtt majdnem minden alkalmazás más formátummal dolgozott, így ha egy adatot más alkalmazással szerettünk volna feldolgoztatni, értelmezőt kellett írni minden egyes szoftverhez. Azonban

[5] eXtensive Stylesheet Language Transformation
[6] eXtensive Stylesheet Language Family

a szemantikus web koncepciójához nem elég, mivel az alkalmazások közti XML alapú kommunikáció megteremtéséhez egyeztetni kell az átvitt információ szemantikáját, hogy egy XML dokumentumban megadott adaton minden szoftver ugyanazt értse.

A szemantikus web elterjedéséhez viszont alapvető, hogy az adatok közzétételekor minden fél ugyanazt értse egy adaton. Ennek érdekében jött létre az RDF nyelv.

2.2. RDF

Az RDF[7] lényege, hogy tetszőleges *erőforrásokat* leírhassunk metaadatokkal. Erőforrásnak tekinthető bármi, ami URI-val azonosítható. Az URI-kat már említettem az XML névterek kapcsán, most lássunk néhány példát, hogyan is néznek ki:

```
http://www.inf.unideb.hu
file:///home/user/pelda.pdf
ftp://ftp.pelda.hu/uri
```

Az URI-k legtöbbször interneten elérhető erőforrásokat azonosítanak, viszont nem minden esetben. URI-t lehet rendelni a tárgyi világ objektumaihoz is, például egy szobában lévő asztalhoz is, a lényeg, hogy az azonosító egyedi legyen.

Egy URI rendelkezhet egy úgynevezett *erőforrásrész-azonosító*val, ami lehetővé teszi egy másodlagos erőforrás közvetett azonosítását egy elsődleges erőforrásra hivatkozáson keresztül. Ez az azonosító a # jeltől az URI végéig tart. Egy URI lehet *abszolút URI*, vagy *URI hivatkozás*. Az abszolút URI nem tartalmaz erőforrásrész azonosítót, hogy lehessen *bázis URI*-ként is használni. Az URI hivatkozás URI, vagy úgynevezett *relatív hivatkozás*. Egy relatív hivatkozás egy adott környezetben értelmezett, feloldani a már előbb említett bázis URI segítségével lehet. Bázis URI-t definiálhatunk egy dokumentumban, ha ezt nem tesszük meg, akkor a bázis a tartalmazó objektum URI-ja lesz, vagy ha ilyen nincs, akkor az az URI lesz, ahol az adott objektum elérhető. Az URI-k a relatív hivatkozástól eltekintve mindig a használat környezetétől függetlenül azonosítanak erőforrásokat. Benkő Tamás, Szerendi Péter és Lukácsy Gergely *A szemantikus világháló elmélete és gyakorlata* című könyvüben[13] a *relatív URI* kifejezést használják a relatív hivatkozás helyett, de ezt a szabvány nem használja.

RDF segítségével úgy közlünk metaadatokat, hogy egy URI-val ellátott erőforrásokat tulajdonságok segítségével más erőforrásokkal vagy literálokkal kötjük össze. Az RDF adatmodell egy halmazelméleti modell, amely négy halmazt definiál, ezek segítségével írhatunk le metaadatokat:

[7] Resource Descripton Framework – Erőforrás Leíró Keretrendszer

- Erőforrások (Resources) halmaza: A halmaz elemeit erőforrásoknak hívjuk, minden egyes erőforrást URI-k azonosítanak.
- Tulajdonságok (Properties) halmaza: Erőforrásokhoz kapcsolható jellemzők, amik valójában erőforrások, tehát őket is URI-k azonosítják. A halmaz elemeit tulajdonságoknak nevezzük.
- Literálok (Literals) halmaza: Elemei literálok, azaz karaktersorozatok.
- Kijelentések (Statements) halmaza: A halmaz elemei kijelentések (hármasok), amik alanyból, állítmányból és tárgyból állnak. Az alany lehet *erőforrás*, az állítmány *tulajdonság*, a tárgy lehet *erőforrás*, vagy *literál*. Egy hármas azt jelenti, hogy az alany és a tárgy az állítmány által jelölt viszonyban van egymással, tehát binér relációkat írhatunk le velük.

2.2.1. Szintaxisok

Ennek az adatmodellnek a *szemantiká*ja az, hogy a kijelentések igazak, viszont a *szintaxis*ról nem mond semmit. A szintaxist három szabványos adatmodell reprezentáció adja meg:

- Hármasok halmaza
- Címkézett irányított gráf
- XML

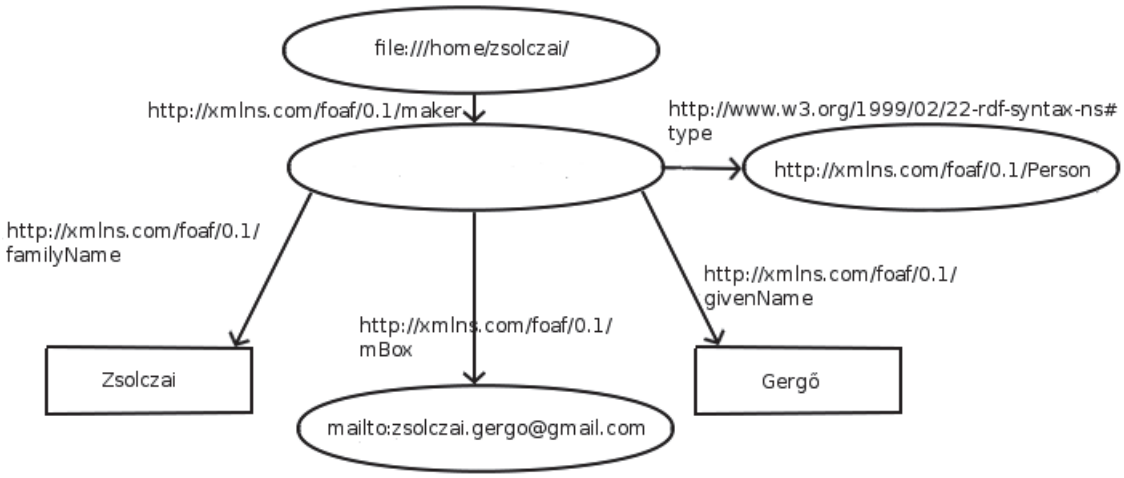
Ezek közül az XML a legelterjedtebb, mert ez biztosítja a hordozhatóságot és a könnyű gépi feldolgozást. Példa az RDF XML szintaxisára:

```
<?xml version="1.0" encoding="UTF-8"?>

<rdf:RDF
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:foaf="http://xmlns.com/foaf/0.1/">
    <rdf:Description rdf:about="file:///home/zsolczai/">
        <foaf:maker>
            <foaf:Person>
                <foaf:givenName>Gergő</foaf:givenName>
                <foaf:familyName>Zsolczai</foaf:familyName>
                <foaf:mbox
        rdf:resource="mailto:zsolczai.gergo@gmail.com"/>
            </foaf:Person>
        </foaf:maker>
```

```
    </rdf:Description>
</rdf:RDF>
```

A fenti példában lévő `foaf:Person` elem egy úgynevezett *köztes erőforrás*, amit arra szoktunk bevezetni, hogy adatainknak további struktúrát biztostsunk. Emellett binárisnál magasabb fokú relációk binárissá leképezéséhez is használhatjuk a köztes erőforrásokat, mivel RDF segítségével csak binér relációkat írhatunk le. Az előző példa gráf reprezentációja talán egy kicsit jobban szemlélteti a köztes erőforrás használatát:



2.1. ábra. Egyszerű RDF gráf

Az erőforrásokat ellipszis jelöli, a literálokat téglalap, a tulajdonságokat irányított címkézett élek. A teljesség kedvéért bemutatom példám RDF hármas halmaz reprezentációját is:

```
{[file:///home/zsolczai/],[http://xmlns.com/foaf/0.1/maker],
    [http://xmlns.com/foaf/0.1/Person]}
{[http://xmlns.com/foaf/0.1/Person],
    [http://xmlns.com/foaf/0.1/givenName],"Gergő"}
{[http://xmlns.com/foaf/0.1/Person],
    [http://xmlns.com/foaf/0.1/familyName],"Zsolczai"}
{[http://xmlns.com/foaf/0.1/Person],
    [http://xmlns.com/foaf/0.1/mbox],
    [mailto:zsolczai.gergo@gmail.com]}
```

Láthatjuk, hogy a három féle reprezentáció közül ez a legrövidebb, viszont a gráf emberi szemmel átláthatóbb, gépi feldolgozás szempontjából pedig az XML a hatékonyabb.

Az RDF XML leírása a szokásos fejléccel kezdődik, ami jelzi, hogy adott verziószámú XML szintaktikát használó dokumentumról van szó. Gyökéreleme az `rdf:RDF` jelzi, hogy itt

kezdődik az RDF leírás. Az `rdf` prefixhez rendelt névtér az RDF specifikáció része. A másik deklaráció a FOAF[8] névtere, ami egy olyan szabványos szókészlet, ami emberek és kapcsolataik leírására szolgál [1]. Az `rdf:Description` elem jelzi, hogy az `rdf:about` attribútum értékeként megadott erőforrás leírása következik.

2.2.2. RDF osztályok

RDF-ben lehetőségünk van osztályok „példányosítására" az objektum-orientált világhoz hasonló módon, azzal a különbséggel, hogy itt nem új példányokat hozunk létre, csupán kijelentjük valamiről, hogy az egy bizonyos osztály példánya. Saját osztály definiálására maga az RDF nem képes, ezt a később ismertetett RDF sémákkal tehetjük meg. Ez történik például az előző XML szintaxisú RDF példa leírásban is, ahol kijelentem, hogy Zsolczai Gergő a `foaf:Person` osztály példánya, ahol a `foaf` előtaghoz hozzárendelem a `http://xmlns.com/foaf/0.1/` névteret. Az előbb bemutatott példa valójában a példányosítás rövid alakja, teljes alakjához az `rdf:type` tulajdonságot használjuk:

```
<rdf:Description>
    <rdf:type resource="http://xmlns.com/foaf/0.1/Person">
        <foaf:givenName>Gergő</foaf:givenName>
        <foaf:familyName>Zsolczai</foaf:familyName>
        <foaf:mbox
            rdf:resource="mailto:zsolczai.gergo@gmail.com"/>
    </rdf:type>
</rdf:Description>
```

Egy erőforrás több osztály példánya is lehet egyszerre, viszont rövidített alakot csak egyhez használhatunk.

2.2.3. Tárgyiasított kijelentések

Mivel RDF kijelentéseket mindenki tehet, mint ahogyan a világhálón is bárki bármilyen információt közzétehet, fontos, hogy a kijelentésekhez hozzárendeljük, kihez tartozik, esetleg mikor történt a kijelentés, hiszen nem feltétlenül igaz minden kijelentés. Az erre szolgáló eszközt az RDF-ben *magasabb rendű kijelentések*nek, vagyis kijelentésekről szóló kijelentéseknek nevezzük. Mivel az RDF kijelentés tárgyának erőforrásnak kell lennie, ezért kijelentésünket egy speciális erőforrással, úgynevezett *reifikált* (tárgyiasított) *kijelentés*sel írjuk le, ami valójában az `rdf:Statement` osztály példánya. Példányosításkor a reifikált kijelentés alanyát,

[8] Friend of a Friend

állítmányát és tárgyát rendre az rdf:subject, rdf:predicate, rdf:object tulajdonságokkal adhatjuk meg. Ha kész van a reifikált kijelentésünk, már csak annyi van hátra, hogy a hozzá tartozó URI-t hozzárendeljük egy kijelentéshez, mint a kijelentés tárgyát. Ennek a módszernek viszont van egy hiányossága, ez pedig az, hogy a tárgyiasított állítást közvetlenül nem tudjuk összekapcsolni azzal az állítással, amire vonatkozik.

2.2.4. Konténerek

Dolgok összességének leírására az RDF konténereket alkalmaz. Konténerek leírására az RDF beépített osztályokat és tulajdonságokat kínál fel. A beépített osztályok a következők:

- Bag (rdf:Bag): erőforrások vagy literálok olyan csoportját reprezentálja, amelyben megengedett az ismétlődés és lényegtelen a tagok sorrendje.
- Sequence (rdf:Seq): erőforrások vagy literálok olyan csoportját reprezentálja, amelyben megengedett az ismétlődés és lényeges a tagok sorrendje. Tekinthetjük egy rendezett Bagnek is.
- Alternative (rdf:Alt) erőforrások vagy literálok olyan csoportját reprezentálja, amelyben megengedett az ismétlődés és lényegtelen a tagok sorrendje, és a tagok bizonyos szempontból egyenértékűek, felcserélhetőek egymással. A másik két konténerrel ellentétben ez nem lehet üres, legalább egy elemnek kell benne szerepelnie és ez az elem a konténer alapértelmezett eleme.

Mivel RDF-fel csak kijelentéseket tehetünk, az osztályhoz hasonló módon nem hozunk létre új konténert, hanem csak kijelentjük egy erőforrásról az rdf:type tulajdonság segítségével, hogy az konténer. Egy konténer bármilyen elemet tartalmazhat és egy konténerről bármilyen kijelentést tehetünk. Kijelentést úgy tehetünk egy konténerről, hogy azt olyan tulajdonságelemekkel látjuk el, amik nem a konténer elemeit jelzik. Elemeket az rdf:_n tulajdonsággal adhatunk meg, ahol n természetes szám. Az RDF konténerek kezelése azért hasonlít annyira az osztályok kezeléséhez, mert az RDF-ben a Bag, Sequence és az Alternative valójában osztályok. Példa konténer definiálására:

```
<rdf:Description rdf:ID="Bevásárlókosár">
    <rdf:type rdf:resource=
        "http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag"/>
    <t:cim rdf:resource="http://www.pedla.hu/webshop"/>
    <rdf:_1 rdf:resource=
        "http://www.pelda.hu/konyv/Szemweb"/>
```

```
    <rdf:_2 rdf:resource=
        "http://www.pelda.hu/szgep/optikai_eger"/>
</rdf:Bag>
```

Rövid alakban:

```
<rdf:Bag rdf:ID="Bevásárlókosár">
    <t:cim rdf:resource="http://www.pedla.hu/webshop"/>
    <rdf:_1 rdf:resource=
        "http://www.pelda.hu/konyv/Szemweb"/>
    <rdf:_2 rdf:resource=
        "http://www.pelda.hu/szgep/optikai_eger"/>
</rdf:Bag>
```

Az elemek felsorolásánál az elemek explicit beszámozása helyett használhatjuk az `rdf:li` jelölést is, az RDF állomány feldolgozásakor a `li` helyére automatikusan behelyettesítődik a megfelelő `_1`, `_2`, stb.

2.2.5. Kollekciók

A konténerekkel ellentétben a kollekciók csak az általunk felsorolt elemeket tartalmazzák. Egy RDF kollekció valójában egy lista, vagyis az `rdf:List` osztály példánya. A lista első elemét az `rdf:first`, a többit az `rdf:rest` tulajdonság jelzi. Az `rdf:rest` értéke egy újabb `rdf:List` erőforrás lehet, vagy lista vége esetén a `http://www.w3.org/1999/02/22-rdf-syntax-ns#nil` erőforrás. Példa lista használatára:

```
<rdf:Description rdf:about="http://www.peldahonlap.hu">
    <foaf:maker>
        <rdf:List>
            <rdf:first>
                <foaf:Person>
                    <foaf:name>Gipsz Jakab</foaf:name>
                </foaf:Person>
            </rdf:first>
            <rdf:rest>
                <rdf:List>
                    <rdf:first>
                        <foaf:Person>
                            <foaf:name>Wincs Eszter</foaf:name>
                        </foaf:Person>
                    </rdf:first>
                    <rdf:rest rdf:resource=
```

```
                    "www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                </rdf:List>
            </rdf:rest>
        </rdf:List>
    </foaf:maker>
</rdf:Descripcion>
```

2.2.6. Adattípusok

Literálok használatánál gyakran felmerülhet igény arra, hogy típussal lássuk el őket, hogy a feldolgozó alkalmazás egy literált ne csak karaktersorozatként értelmezzen, hanem számként, esetleg dátumként, stb. Literál típusának megadására az RDF az `rdf:datatype` attribútumot használja, melynek értéke tetszőleges URI lehet. Tetszőleges, mert az RDF nem törődik vele, hogy az az URI mindenkinek ugyanazt a típust jelöli-e, ezzel a feldolgozó alkalmazásnak kell törődnie. Ezen URI-k megadásakor ajánlott a már említett W3C XML Schema által ajánlott típusokat használni az egyértelműség biztosítására. Példa adattípus használatára:

```
<rdf:Description rdf:about="http://www.peldahonlap.hu">
    <k:készült rdf:datatype="xs:date">2011-03-30</k:készült>
</rdf:Description>
```

Az XML séma által támogatott további típusokról a W3X XML Schema specifikációjában olvashatunk [11].

2.2.7. Következtetés

Ahogy azt a fejezet elején is leírtam, a szemantikus világháló ötlete a metainformációk közlése mellett az információk alapján történő következtetésre is épül. Ezen következtetéseket viszont az eddig ismertetett módszerekkel nem lehet elvégezni. Ehhez szükség van olyan eszközökre, amelyekkel a tulajdonságok és az osztályok közötti kapcsolatokat tudjuk ismertetni, és ha szükséges, létre is hozhassunk új tulajdonságokat és osztályokat. A fejezet első példájával elmagyarázva, létrehozhatunk egy *Macska* és egy *Emlős* osztályt, és leírhatjuk a köztük lévő tartalmazási viszonyt, hogy a *Macska* egy speciális *Emlős* osztály, más szóval annak leszármazottja. Emellett arra is szükség van, hogy kijelenthessük, hogy a *Macska* és a *Cica* osztály ekvivalens, tehát kezeljük a szinonimákat. Ezeket az eszközöket az RDF sémák biztosítják.

2.2.8. RDF sémák

Az RDF sémák ismertetését azzal kezdeném, hogyan is lehet kijelenteni egy erőforrásról, hogy az osztály. Az RDF működéséből adódóan ebben az eset-

ben sem hozzuk létre az új osztályt. Ezt úgy tehetjük meg, hogy az erőforrás típusának a `http://www.w3.org/2000/01/rdf-schema#Class` értéket adjuk, vagyis az `rdf:type` értékeként az `rdfs:Class` erőforrást tüntetjük fel, ahol `xmlns:rdf="www.w3.org/1999/02/22-rdf-syntax-ns#"` és `xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"`. Lényegében kijelentjük, hogy az adott erőforrás az `rdfs:Class` osztály példánya. Az osztály-alosztály viszonyt az `rdfs:subClassOf` tulajdonsággal írhatjuk le. Az RDF séma a többszörös öröklődés elvét vallja, vagyis egy osztálynak tetszőleges számú szülőosztálya lehet, emellett tetszőleges számú gyermek osztálya lehet, és az öröklődés tranzitív. Példa osztály és tulajdonság definiálására és a kapcsolatok ismertetésére:

```
<?xml version="1.0" encoding="UTF-8"?>

<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:base="hppt://www.pelda.hu"
        xmlns:xs="http://www.w3.org/2001/XMLSchema#">

    <rdfs:Class rdf:ID="Emlős">
        <rdfs:comment>Emlősök osztálya</rdfs:comment>
    </rdfs:Class>

    <rdfs:Class rdf:ID="Macska">
        <rdfs:subClassOf
            rdf:resource="#Emlős"/>
        <rdfs:comment>Macskák osztálya</rdfs:comment>
    </rdfs:Class>

    <rdf:Property rdf:ID="születésDátuma">
        <rdfs:domain rdf:resource="#Emlős"/>
        <rdfs:range rdf:resource=
            "http://www.w3.org/2001/XMLSchema#date"/>
    </rdf:Property>

    <rdf:Description rdf:ID="Kormos">
        <rdf:type rdf:resource="#Macska"/>
        <születésDátuma rdf:datatype=
            "xs:date">2000-01-01</születésDátuma>
    </rdf:Description>

    <Macska rdf:ID="Garfield">
        <születésDátuma rdf:datatype=
```

```
            "xs:date">1978-06-19</születésDátuma>
    </Macska>
</rdf:RDF>
```

A fenti példában használom az `rdfs:comment` tulajdonságot, ami arra szolgál, hogy ember számára érthető rövid szöveges leírást adjunk. Általunk létrehozott osztály példányosítására is láthatunk példát rövid és hosszú alakban is. Láthatjuk, hogy tulajdonságot hasonló módon lehet megadni, mint osztályt, a különbség az, hogy a tulajdonság az `rdf:Property` osztály példánya. Tulajdonságok esetén is definiálhatunk ős-leszármazott viszonyt, erre a `rdfs:subPropertyOf` tulajdonság szolgál. Ebben az esetben is tranzitív a viszony. Az `rdfs:domain` segítségével az adott tulajdonság értelmezési tartományát adhatjuk meg, az `rdfs:range` tulajdonsággal értékkészletet adhatunk meg. Egy tulajdonság értelmezési tartománya és értékkészlete az `rdfs:domain` és `rdfs:range` tulajdonságok értékeként megadott osztályok vagy adattípusok. RDF kijelentés esetén ezen tulajdonságok alanyának és tárgyának az adott osztályok példányait kell megadnunk. A tulajdonságok alapértelmezett értelmezési tartománya és értékkészlete az `rdfs:Resource`, ez annyit tesz, hogy ha ezekre nincs megadva explicit megszorítás, akkor az adott tulajdonság tetszőleges osztályba tartozó erőforrásra alkalmazható. Ha több értelmezési tartományt adunk meg, akkor az értelmezési tartomány a megadott osztályok metszete lesz. Ugyanez érvényes az értékkészletre is. Az RDF séma által kínált további lehetőségekről az RDF séma specifikációjában olvashatunk bővebben [5].

2.2.9. Tulajdonságközpontúság

Bár az RDF-nek is van osztály és példány fogalma, az objektum-orientált szemlélettel ellentétben az RDF egy *tulajdonságközpontú* nyelv. Ez annyit tesz, hogy a tulajdonságokat csak „futási időben” rendeljük hozzá az erőforrásokhoz, vagyis az osztály definiálásakor még nem adjuk meg, hogy azok példányai milyen attribútumokkal fognak rendelkezni. Ez nagyobb rugalmasságot biztosít, mert egy létező osztály példányához bárki bármilyen tulajdonságot hozzárendelhet, szemben az objektum-orientált rendszerekkel, ahol a példány csak és kizárólag azokkal a tulajdonságokkal rendelkezhet, amiket már előre definiáltak az osztályában. A tulajdonságközpontúság közel áll az emberi gondolkodáshoz, mert ezzel a módszerrel egy rendszer képes az új ismeretek dinamikus rögzítésére, ezért alkalmasabb a világháló leírására is, mint a statikusabb objektum-orientált szemlélet.

2.2.10. RDF összefoglalás

A szemantikus világháló megalapozásának második legnagyobb lépése az XML után az RDF megjelenése. Bemutattam az RDF leggyakrabban használt szintaxisait, legfőképpen az XML szintaxist, ami megfelelő alapja lehet a szemantikus web elterjedésének. Ismertettem, hogyan, milyen szerkezeteket használva lehet felépíteni egy RDF leírást. Kitértem arra is, hogy RDF leírást bárki készíthet, ezért az RDF rendelkezik egy olyan szerkezettel, amivel jelölhetjük, kihez tartoznak az adott kijelentések, viszont ebből adódik egy jelenleg még megoldaltlan probléma, mégpedig az, hogy semmi sem garantálja, hogy egy állítás igaz. Ez egyébként jellemző a web mai formájára is, bár már vannak sikeresnek mondható eredmények a hamis tartalmak kiszűrésére, például a Google PageRank algoritmusa kisebb prioritást rendel a rosszabb hírű oldalakhoz. A konténerek, kollekciók használatával struktúráltabbá tehetjük közölni kívánt adatainkat, az adattípusok segítségével pedig segíthetjük a feldolgozó alkalmazásokat. Bemutattam, hogyan lehet megvalósítani az RDF sémák segítségével, hogy az RDF által közölt információk alapján a feldolgozók következtetni tudjanak. Az RDF keretrendszer ismertetését a típusközpontúság kifejtésével zártam. Megemlíteném még az OWL webontológia nyelvet, ami az RDF sémáknál komplexebb módon teszi lehetővé, hogy következtetéseket vonjunk le a metaadatok alapján.

Szakdolgozatomat egy RDF alapú szótár, a DOAP tárgyalásával folytatom. Szemantikus Web fejezetem utolsó szakasza fontos tudnivalókat tartalmaz Apache Maven bővítményem funkcionalitásának megértéséhez, hiszen pluginom segítségével DOAP dokumentumot állíthatunk elő.

2.3. DOAP

A DOAP[9] egy XML/RDF alapú szókészlet, melynek segítségével (elsősorban nyílt forráskódú) szofterprojekteket írhatunk le. A projekt nevét az RDF keretrendszernél már említett FOAF ihlette. A DOAP együttműködést nyújt más népszerű web metaadat projektekkel, mint például az RSS, a FOAF és a Dublin Core. Az ily módon elkészített dokumentumban megadhatjuk szoftver projektünk nemzetköziesített leírását, a projekthez kapcsolódó személyeket és a webes erőforrásokat is. A DOAP támogatja projektek szoftverkönyvtárakhoz rendelését, a szoftverkönyvtárak közötti adatcserét és tárolók automatikus konfigurációját.

[9]Description of a Project

2.3.1. Szintaxis

Edd Dumbill – a projekt vezetője – elgondolása az volt, hogy amellett, hogy a DOAP leírásokat számítógép által könnyen feldolgozhatóra tervezzék, a szótárak „emberi fogyasztásra" is alkalmasak legyenek. Amint azt már az előző fejezetekben tárgyaltam, ennek megoldására a ma ismert legmegfelelőbb technológia az XML, vagy méginkább az XML szintaxist használó RDF. A szemantikus web leírónyelveként az RDF már igen elterjedt, néhány ember azonban kerüli ezt a megoldást azzal a magyarázattal, hogy az RDF is XML formátum, és igazi előnyét nem tudjuk kihasználni, amíg nem RDF-et támogató feldolgozó eszközt használunk, ezért ők a tisztán XML alapú reprezentáció hívei.

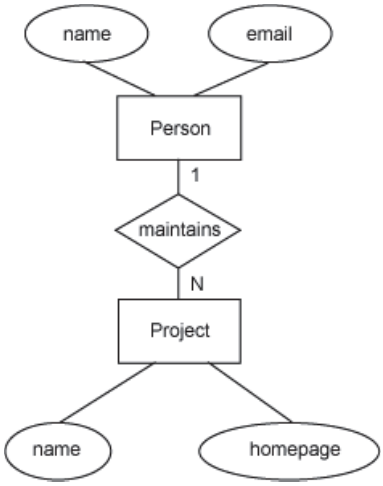
Az efféle szerializációnak is megvannak a maga nehézségei. A dokumentum struktúra megválasztásához több sémanyelv is igénybe vehető, amiknek különböző a kifejezőereje és az eszköztámogatottsága. A DTD[10] annak ellenére, hogy még mindig nagyon elterjedt, elég primitív, szegényes az eszköztámogatottsága. A W3C XML Schema már sokkal rugalmasabb, de szintaxisa és annak használata kissé nehézkes. A RELAX NG-t viszont Edd Dumbill egy ígéretes újoncnak tartja, ami könnyebben megérthető, mint a W3C XML Schema. Rendelkezik XML és nem-XML szintaxissal is, eszközöket nyújt az ezek közti konverzió megvalósítására, továbbá könnyen konvertálható a fentebb említett sémanyelvekre is. A nem-XML szintaxis jóval tömörebb és átláthatóbb, így szerkesztőprogram nélkül is könnyű megírni. A DOAP tisztán XML alapú reprezentációjához ezért a RELAX NG a legkézenfekvőbb megoldás, viszont még mindig fennáll egy probléma, ami az XML jellegéből adódik. Habár a szintaktika jól definiált, az XML nem mond semmit az elemek szemantikájáról. Az RDF séma megengedi, hogy például egy szoftver projekt tulajdonosa a `creator` Dublin Core tulajdonság altulajdonsága legyen, ezért bármilyen RDF alkalmazás, ami tudja kezelni a Dublin Core-t, a DOAP adatkezelésének legalább az alapjaival tisztában van. Ezzel szemben egy tiszta XML dokumentum nem jelent semmit egy olyan alkalmazásnak, amibe explicit módon nincs belekódolva, hogyan dolgozza fel a DOAP névteret, még ha rendelkezik is a megfelelő sémával.

2.3.2. Adatok szerkezete

A szoftverkönyvtár weboldalak, mint például a GNOME, OMF, Sourceforge nem feltétlenül ugyanazokat a metaadat elemeket használja. Ezen oldalak közül a legtöbben egy egyszerű modellt alkalmaznak, ahol a projekt egy önálló egyed, metaadatai pedig ennek az egyednek egyszerű tulajdonságai. Összetett tulajdonságot többféleképpen is meg lehet adni. Egyik példa, ha a tulajdonság tartománya az emberek halmaza. Egy embert ugyebár nem lehet csupán a nevével

[10]Document Type Definition

azonosítani. A 2.2 ábrán a példámhoz tartozó részleges egyed-kapcsolati diagramot láthatjuk, melynek forrása Edd Dumbill *XML Watch: Describe open source projects with XML* cikkének második része [8].



2.2. ábra. Egyed-kapcsolati diagram

Mivel az összetett tulajdonságokat diagramon könnyen reprezentálhatjuk kapcsolattal, ezért a kihívás nem a modellezésben, hanem inkább abban rejlik, hogy a DOAP-ot úgy tervezzük meg, hogy könnyen előállítható és feldolgozható legyen.

Ahhoz, hogy hatékonyan manipuláljuk a szótárunkban leírt adatokat, ki kell jelölnünk legalább egy tulajdonságot projektünk azonosítására. Ez ahhoz hasonló, mint mikor egy adatbázistáblában lévő oszlopot kijelölünk elsődleges kulcsnak, azzal a különbséggel, hogy itt nem elég egy lokálisan egyedi kulcsot választani, ha DOAP-unkat közzé szeretnénk tenni a weben. A DOAP egyik alapelve, hogy decentralizált, vagyis a leírásokat anélkül el lehet készíteni, és meg lehet osztani, hogy egy bizonyos weboldalon regisztrálni kellene.

Ahogy azt már az RDF tárgyalásánál említettem, a világhálón egy elem globális azonosítására a legmegfelelőbb módszer, ha hozzárendelünk egy URI-t. Mivel a nagyobb szoftver projekthez tartozik weboldal, projektünk azonosítására feltüntethetjük weboldalának címét. Ha csak egy nevet használnánk erre a célra, a világhálón könnyen előfordulhatna duplikáció a projektnevekre vonatkozóan. Projekt weboldal URI-kkal ez a probléma nem áll fenn a DNS rendszernek köszönhetően.

A webcímek erőforrásleíróként használatának viszont hátránya is van, ez pedig a változékonyság. Egy weboldal például már nem lesz elérhető, ha a domain előfizetés lejár, és azt nem hosszabbítják meg. Emellett példa lehet e változékonyság szemléltetésére, ha egy projekt tulajdonosának személye változik és emiatt az erőforrások is változnak – például a projektet felvásárolja egy másik cég –, a tulajdonos domain-t vált, és így tovább, ilyenkor DOAP leírásunk

érvényét veszti. Ennek a problémának a megoldására a DOAP lehetővé teszi, hogy dokumentumunkban feltüntessünk egy, vagy akár több `old home page` tulajdonságot. Az egyetlen megszorítás, hogy az ilyen tulajdonságokban megadott URI-kat más projektek soha nem használhatják. Ha egy projekt weboldalának címe megváltozik, és a projekt leírására több független DOAP létezik, de csak az egyikben szerepel az új cím `home page` tulajdonság értékeként, viszont tartalmaz egy `old home page` tulajdonságot a régi URI-val, a feldolgozók felismerik, hogy ugyanarról a projektről van szó. Ezt a módszert alkalmazzák egyébként FOAF projektben is.

DOAP leírás készítésekor alaposan fontoljuk meg, hogy mely tulajdonságok értékénél használunk literálokat, és melyeknél URI-t. Ott, ahol fontos, hogy egy metaadat plusz információt hordozzon, esetleg számítógép által ellenőrizhető legyen, használjunk URI-t, például általánosan ismert licenszek feltüntetésekor használjuk azok URI-ját, saját licensz megadásakor webtárhelyünkön hozzunk létre egy külön RDF dokumentumot licenszünk ismertetésére, és rendeljünk hozzá egy URI-t. Ilyenkor persze szintén fennállhat az a probléma, hogy az URI megváltozik, és a régi URI feletti irányítást elveszítjük. Erre Edd Dumbill *XML Watch: Describe open source projects with XML* cikkének második részében [8] említett megoldása: használjuk a *purl.org*-ot, vagy ehhez hasonló szolgáltatást, ami garanciát ad az ott regisztrált URI-nk élettartamára.

A továbbiakban példákkal illusztrálom, hogyan is néz ki egy RDF sémával reprezentált DOAP. Következő példámat Edd Dumbill *XML Watch: Describe open source projects with XML* cikkének harmadik részéből [9] vettem, ami magáról a DOAP projektről szóló DOAP leírást mutatja be:

```
<Project xmlns="http://usefulinc.com/ns/doap#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/">

    <name>DOAP</name>
    <homepage rdf:resource="http://usefulinc.com/doap" />
    <created>2004-05-04</created>
    <shortdesc xml:lang="en">
        Tools and vocabulary for describing community-based
        software projects.
    </shortdesc>
    <description xml:lang="en">
        DOAP (Description of a Project) is an RDF vocabulary
        and associated set of tools for describing community-
        based software projects. It is intended to be an
        interchange vocabulary for software directory sites,
        and to allow the decentralized expression of
        involvement in a project.
```

```
    </description>
    <maintainer>
        <foaf:Person>
            <foaf:name>Edd Dumbill</foaf:name>
            <foaf:homepage
                rdf:resource="http://usefulinc.com/edd" />
        </foaf:Person>
    </maintainer>
</Project>
```

Fontos szabályok DOAP írásakor:

- Az osztályok nevét nagy kezdőbetűvel írjuk, mint ahogy az látható a `Project`-nél és a `Person`-nál, ez egy RDF konvenció. A tulajdonságok nevét csupa kisbetűvel írjuk.
- Egy DOAP dokumentum legkülső eleme a `Project`. Az RDF szintaxis [6] megengedi az `rdf:RDF` gyökérelem elhagyását, amennyiben a leírás megadható egy külső elemmel.
- A DOAP névterének URI-ja `http://usefulinc.com/ns/doap#`
- Az `xml:lang` attribútum a tulajdonságok értékeként megjelenő szöveg nyelvét jelöli.

Egy DOAP leírásban három osztály is felhasználható:

- `Project`: a projekteket reprezentáló erőforrás osztálya
- `Version`: a kiadott szoftverek verzióit reprezentáló osztály
- `Repository`: egy verziókezelő rendszer tárolóját reprezentáló példány osztálya

A `Repository`-nak vannak alosztályai is, ezeket a `Project` osztály tárgyalása után mutatom be.

A Project osztály

A `Project` a DOAP főosztálya, minden példány egyedien azonosított a saját weboldalának URI-jával. Emellett ajánlott a projekt leírásban feltűntetni az `old home page` URI-kat is, hogy a `home page` változásakor is lehessen azonosítani a projektet. A projekt lehetséges tulajdonságai rövid magyarázattal:

Tulajdonság	Leírás
name	A projekt neve.
shortname	A projekt rövid neve, amit gyakran fájlneveknél használnak.
homepage	A projekt weboldalának URI-ja, ami csak ehhez a projekthez van rendelve.
old-homepage	A projekt régi weboldalának URI-ja, ami csak ehhez a projekthez van rendelve.
created	A projekt létrehozásának dátuma `YYYY-MM-DD` formátumban.
description	A projekt szöveges leírása.
shortdesc	A projekt rövid szöveges leírása (8-9 szó).
category	Egy URI, ami egy a projekthez rendelt kategóriát azonosít.
wiki	A projekthez rendelt Wiki URI-ja.
bug-database	Egy hibakövető rendszer URI-ja, vagy egy e-mail cím, amire a projekttel kapcsolatos hibákat jelenthetjük.
screenshots	Egy a projekt képernyőképeit tartalmazó weboldal URI-ja.
mailing-list	A projekthez tartozó levelezőlista URI-ja.
programming-language	A projekt implementálására használt nyelv.
os	Az az operációs rendszer, amin a projekt fut (platformfüggetlenség esetén elhagyhatjuk).
license	A projekt licenszének URI-ja.
download-page	Az az URI, amiről a projekt letölthető.
download-mirror	Egy letöltési tükör szerver címe.
repository	Egy `doap:Repository` példány, ami a projekthez tartozó forráskód tárolót adja meg.
release	Egy `doap:Version` példány, ami a szoftver projekt aktuális kiadását írja le.
maintainer	Egy `foaf:Person` példány, ami a projekt tulajdonosát vagy vezetőjét adja meg.
developer	Egy `foaf:Person` példány, ami a projekt fejlesztőjét adja meg.
documenter	Egy `foaf:Person` példány, ami a projekt dokumentáló munkatársát adja meg.
translator	Egy `foaf:Person` példány, ami a projekt fordító munkatársát adja meg.
helper	Egy `foaf:Person` példány, ami a projekt olyan munkatársát adja meg, akiket más tulajdonságokkal nem lehet feltüntetni.

A Repository osztályok

A Repository osztályt, vagyis inkább a négy alosztályát forráskód tárházak leírására használhatjuk. Az alosztályok a következők: *Subversion*, *BitKeeper*, *CVS*, és *GNU Arch*, melyek bizonyos ma elterjedten használt verziókezelő rendszereknek felelnek meg. Az egyes Repository alosztályokhoz tartozó tulajdonságokat szemléltető táblázat:

Tulajdonság	Leírás	SVNRep.	BKRep.	CVSRep.	ArchRep.
anon-root	A névtelen módon elérhető tároló elérési útja			*	
module	A forráskód tárolóban lévő modul neve			*	*
browse	A tárolóhoz webböngésző interfészének URL-je	*	*	*	
location	Az arhívum URI-ja	*	*		*

E rendszerek használatára Edd Dumbill *XML Watch: Describe open source projects with XML* cikkének harmadik részéből [9] választottam példákat:
Subversion:

```
<SVNRepository>
    <location
        rdf:resource="http://svn.usefulinc.com/svn/repos/trunk/doap/"/>
    <browse rdf:resource=
        "http://svn.usefulinc.com/cgi-bin/viewcvs.cgi/trunk/doap/"/>
</SVNRepository>
```

BitKeeper:

```
<BKRepository>
    <location rdf:resource="http://linux.bkbits.net/linux-2.6"/>
    <browse rdf:resource="http://linux.bkbits.net:8080/linux-2.6"/>
</BKRepository>
```

CVS:

```
<CVSRepository>
    <anon-root>:pserver:anonymous@anoncvs.gnome.org:/cvs/gnome</anon-root>
    <module>epiphany</module>
    <browse rdf:resource="http://cvs.gnome.org/viewcvs/epiphany/"/>
</CVSRepository>
```

GNU Arch:

```
<ArchRepository>
    <location rdf:resource="http://www.gnome.org/~jdub/arch"/>
    <module>jdub@perkypants.org--projects/planet--devel--0.0</module>
</ArchRepository>
```

A Version osztály

A verziókövetés nem része a DOAP-nak, így új kiadás esetén az aktuálishoz külön DOAP-ot kell készítenünk. A Version osztály reprezentálja a szoftver kiadásait. A Version osztály tulajdonságai és azok lehetséges értékei:

- A `branch` tulajdonság értéke egy karakterlánc, ami a verzió ágát jelzi, mint például `stable`, `unstable`, `gnome24`, vagy `gnome26`.
- A `name` tulajdonság értéke egy kiadásnév, mint például Ubuntu 10.04-nél *Lucid Lynx*.
- A `created` tulajdonság értéke a kiadás dátuma `YYYY-MM-DD` formátumban.
- A `revision` tulajdonság értéke A kiadás verziószáma, mint például 1.0.

Példa egy Ubuntu kiadás DOAP leírására:

```
<Version>
    <branch>stable</branch>
    <name>Maverick Meerkat</name>
    <revision>10.10</revision>
    <created>2010-10-10</created>
</Version>
```

Minden projektnek lehet több aktuális kiadása, ezért van szükség a `branch` tulajdonságra. Gyakori az olyan eset is, hogy egy projekt rendelkezik egy stabil ággal, miközben kiadnak hozzá egy instabilt is új funkcionalitások tesztelésére.

2.3.3. DOAP séma

A DOAP sémában találhatók a DOAP osztályok és a tulajdonságok formális definíciói. RDF sémaként írták és kölcsönvettek hozzá egy tulajdonságot az OWL ontológia nyelvből, hogy az azonosító tulajdonságokat jelölje (amiket az OWL nyelvben inverz funkcionális tulajdonságoknak neveznek).

A DOAP-ot tiszta XML dokumentumként is fel lehet dolgozni, viszont ha már RDF reprezentációt választottunk DOAP-unkhoz, ajánlott RDF-ként is feldolgozni. Ehhez számos eszköz rendelkezésünkre áll. Viszont előnye is van annak, hogy DOAP-unk XML szintaxissal rendelkezik, mégpedig az, hogy ha készíthetünk hozzá egy XSLT stíluslapot, amellyel könnyen olvasható HTML formátumba alakíthatjuk.

2.3.4. DOAP összefoglalás

Szemantikus Web fejezetemet a DOAP szókészlet ismertetésével zárom, mely segítségével szoftver projektekről készíthetünk XML/RDF alapú leírásokat, azzal a céllal, hogy közzétehessük azt a szemantikus világhálón. Bemutattam, hogy a DOAP milyen eszközöket biztosít szoftverünk rövid ismertetésére, a hozzá kapcsolódó licenszek leírására, a verziószám jelzésére, és még néhány fontos dologra, mellyel egyértelműen azonosíthatjuk projektünket.

2.4. Szemantikus Web összefoglalás

Ebben a fejezetben ismertettem, hogy milyen céllal született a szemantikus világháló elgondolás, melynek alapja a weben található erőforrásokhoz metaadatok rendelése, és ezen metaadatok segítségével következtetések levonása. Említettem, hogy metaadatokat manapság már több eltérő technológia is használ, ám ezek többnyire eltérő formátumban jelennek meg. A szemantikus web célja, hogy ezek a metaadatok egy egységes, mindenki számára könnyen feldolgozható formátumban legyenek közzétéve, hiszen a világháló célja, hogy mindenki könnyen információhoz juthasson.

Ilyen formátumot igen elterjedt körben használnak már ma is, ez pedig az XML. Ezen formátum ismertetésére az XML fejezetben bemutattam a szintaxisát, és az XML sémákat, melyekkel megszorításokat tehetünk az XML dokumentumok tartalmára és szerkezetére, a feldolgozó alkalmazások közti félreértések elkerülése végett.

A következő fejezetben bemutattam az XML szintaxissal is rendelkező RDF keretrendszert, mellyel a metaadatok közzétételét nagyon jól meg lehet valósítani az RDF kijelentései segítségével. Az RDF sémákkal pedig biztosíthatjuk, hogy a feldolgozó szoftverek képesek legyenek következtetéseket levonni az RDF dokumentumok által közölt metaadatok alapján.

A fejezetet annak az RDF alapú szókészletnek, a DOAP-nak az ismertetésével zártam, mely formátumú leírás bővítményem által készíthető.

A következő fejezetben bemutatom az Apache Maven-t, azt a projektkezelő rendszert, melyhez a bővítményemet fejlesztettem.

3. fejezet

Maven

A *Maven* az *Apache Software Foundation* (Apache Szoftver Alapítvány) szoftverprojekt kezelő eszköze. Eredetileg a *Jakarta Turbine* projekt build folyamatainak egyszerűsítése céljából készítette az alapítvány, majd továbbfejlesztve egy könnyebb módszert akartak adni projekt információk közzétételére és JAR[1] vagy WAR[2] állományok megosztására. Az eredmény a mai Maven, amivel kezelhetünk bármilyen Java alapú projektet egy POM (Project Object Model) által reprezentált információk alapján. Kezelhetjük projektünk fordítását, dokumentálását és jelentéseket generálhatunk segítségével. Mivel a Maven valójában egy keretrendszer, mindezt pluginokon keresztül valósítja meg. Külön plugin kezeli a fordítás folyamatát, a weboldal generálását a forráskódban lévő dokumentáció és a POM alapján, és még számos funkciót. Generálhatunk vele webszájtot is, amibe API dokumentációt, kódlefedettségi ellenőrzés eredményét és egységteszt eredményeket is belegeneráltathatunk, megkönnyítve ezzel a tesztelés fázisát. A kiadások elkészültével egy paranccsal JAR, vagy WAR állományt generálhatunk, amikbe a Maven automatikusan beágyazza a futtatáshoz szükséges függőségeket, értem ezalatt a már korábban esetleg más gyártó által készített komponenseket. A folytatásban kifejtem, hogyan is néz ki a POM.

3.1. POM

A POM valójában egy XML dokumentum – melynek hagyományosan a **pom.xml** nevet szokták adni –, amiben értelemszerűen XML elemekkel adhatjuk meg projektünk nevét, rövid leírását, esetleg weboldalát, fejlesztői nevét, elérhetőségeit, a projektre vonatkozó licenszeket. Emellett a kezeléshez szükséges pluginokat és a (fordításra, tesztelésre vonatkozó, stb.) függő-

[1] Java Archive File
[2] Web Archive File

ségeket is rögzítenünk kell ebben az állományban.

3.1.1. Kötelező POM elemek

Bár a POM-nak számos eleme van, amikkel projektünkről információt szolgáltathatunk, vagy a vezérléséhez szükséges információkat írhatjuk le, kötelezően csupán a következő elemeket kell tartalmaznia:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>hu.csoport.maven</groupId>
    <artifactId>projektem</artifactId>
    <version>1.0</version>
</project>
```

- A `modelVersion` elem a POM verziószámára vonatkozik. A Maven 2 és 3 jelenleg csak a 4.0.0 verziószámú POM-ot támogatja, tehát ha ezen Maven verziók valamelyikével dolgozunk, a `modelVersion` értékének kötelezően 4.0.0-nak kell lennie.
- A `groupId` egy csoportazonosító, amelynél gakori a fordított domain nevek használata. Az adott projekt azonosítását segíti elő, általában egyedi egy szervezetre vagy projektre nézve.
- Az `artifactId` általában a projekt neve, ami egyértelműen azonosítja a projektet, ezért egy szervezeten belül egyedinek kell lennie, hogy egy tárházban az azonos `groupId` alatt tárolt projektek között is elkerüljük a névütközést.
- Mivel egy projekt általában több verzióval is rendelkezik, és ezeket a verziókat nem dobjuk el az újabb megjelenésekor, szükség van a `version` elemre is, hogy azonosítsuk, a projekt mely verziójáról is van szó.

3.1.2. További POM elemek

Csomagolás

A `packaging` elemmel megadhatjuk projektünk csomagolásának módját, vagyis azt, hogy milyen formátumba szeretnénk csomagolni a kimenetként előállt állományokat. Ezen elem lehetséges értékei: `pom`, `jar`, `maven-plugin`, `ejb`, `war`, `ear`, `rar`, `par`. A `packaging`

elem alapértelmezett értéke `jar`, tehát ha nem adjuk meg ezt az elemet, a Maven `jar` fájlba csomagolja a projektet.

Függőségek

A POM egyik legfontosabb opcionális eleme a `dependencies`, mellyel megadhatjuk projektünk függőségi listáját, egy-egy függőséget a `dependency` elem definiál. A `dependencies` számos `dependency` elemet tartalmazhat. Számos projekt függhet egy másiktól, például ha egy olyan szoftvert fejlesztünk, amihez relációs adatbázis szükséges, függőségként megadhatjuk például az *Oracle JDBC meghajtó*ját, vagy más adatbáziskezelő modult. Függőségkezelésre szükség van akkor is, ha egy komplex szoftverhez csak egy komponenst fejlesztünk. Elég, ha ezeket a függőségeket felsoroljuk a POM-ban és a Maven automatikusan kezeli őket helyettünk. Ha a függőség elérhető a Maven központi tárolójában – ami a legtöbb esetben igaz –, le is tölti nekünk, a projekt csomagolása esetén pedig akár be is illeszti a futáshoz szükséges függőségeket (például JAR, WAR, vagy egyéb) állományunkba. Ha azonban az adott függőség tárgya nem érhető el a központi tárolóban – általában nem nyílt forráskódú tartalomnál, vagy saját fejlesztésű komponensnél –, azt saját tárolónkba kell telepítenünk, mielőtt felhasználnánk.

A `dependency` elem gyermekei lehetnek a már ismertetett projektet azonosító alapelemek, a `groupId`, a `arrifactId`, és a `version`. A továbbiakban bemutatok még néhány elemet, melyekkel a `dependency` gyermekeként további információkat adhatunk meg a függőség tárgyáról:

- A `type` elemmel a csomagolás típusát adhatjuk meg, értékei megegyeznek a `packaging` elem értékeivel.
- A `scope` elemmel írhatjuk le, projektünk mely *életciklusában* van szükség az adott függőségre. Alapértelmezett értéke a `compile`, ami azt jelenti, hogy a függőség tárgyára a fordításnál van szükség. További értékei:
  - A `provided` hasonló a `compile`-hoz, azzal a különbséggel, hogy elvárjuk, hogy futás közben a futtató biztosítsa nekünk az adott függőség tárgyát.
  - A `runtime` érték azt jelenti, hogy az adott függőség csak futás közben szükséges, fordításkor nem.
  - A `test` érték azt jelöli, hogy az adott függőség csak a tesztelésnél szükséges, az alkalmazás normál futása közben nem.

  - A `system` hasonló a `provided`-hez, azzal a különbséggel, hogy mi biztosítjuk a függőség tárgyát.

- A `systemPath` elemet csak akkor használhatjuk, ha a `scope` elem értéke `system`. Ilyenkor egy abszolút elérési utat kell megadnunk ezen elem értékeként. Rendszerspecifikus elérési út esetén ajánlott a tulajdonság használata, mint például a `${java.home}/lib`.

- Az `optional` elemre akkor lehet szükség, ha projektünk szintén egy függőség tárgya.

Kizárások

Kizárásokat akkor használhatunk, ha egy függőségünk tárgyának függőségét nem akarjuk használni projektünknben. A kizárások listáját a `dependency` elemen belül az `exclusions` elemmel adhatjuk meg, ezen belül egy kizárást az `exclusion` elemmel. Egy kizárandó függőséget a `groupId` és az `artifactId` elemek értékeként adhatjuk meg.

Öröklődés

Projektünket származtathatjuk más projektből is, ilyenkor a szülő projekt egyes tulajdonságai (mit például a függőségek, fejlesztők, bővítmény beállítások, stb.) öröklődnek. Bővebben az öröklődésről a [3]-ban olvashatunk.

Aggregáció

A több modulból álló projektet aggregátor projekteknek is nevezzük. A modulokat a `modules` elemben kell felsorolni, egy-egy modul nevét a `module` elem értékeként kell feltüntetni. Példa egy aggregátor projekt POM-jára:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>hu.csoport.maven</groupId>
    <artifactId>projektem</artifactId>
    <version>1.0</version>
    <packaging>pom</packaging>

    <modules>
```

```
        <module>modul1-projekt</module>
        <module>modul2-projekt</module>
        <module>másik-projekt</module>
    </modules>
</project>
```

Build folyamat kezelése

Projektünk build folyamatát a nyitó és záró `build` elemek közötti alelemek megadásával konfigurálhatjuk.

Erre számos eszköz áll rendelkezésünkre, mint például a `defaultGoal` elem, mellyel beállíthatjuk, hogy projektünk felépítése után a futtatás helyett például automatikusan tárolónkba települjön, a `directory` elem, mellyel megadhatjuk, hogy az alapértelmezett `target` könyvtár helyett milyen könyvtárba építse fel projektünket.

Ezen kívül átállíthatjuk a projekt alapértelmezett forráskönyvtárait is, ha a `resources` elemen belül megadunk egy listát `resource` elemekkel – viszont a forráskódot és a teszteseteket tartalmazó könyvtárakat a `build` elemen belül rendre a `sourceDirectory` és a `testSourceDirectory` elemekkel állíthatjuk be –, megadhatjuk a build folyamathoz szükséges bővítményeket a `plugins` elemen belüli `plugin` elemekkel.

További projekt információk

Projektünknek a `name` elemen belül adhatjuk meg egy az `artifactId` elemen belül megadottnál beszédesebb nevet, a `description` elem segítségével egy szöveges leírást közölhetünk projektünkről, az `url` elemen belül pedig projektünk weboldalának címét ismertethetjük. A projekt licenszeit a `licenses` elemen belül felsorolt `license` elemek értékével ismertethetjük. Ha a projekt egy vállalat tulajdonát képezi, a vállalat nevét és weboldalát az `organization` elemben közölhetjük.

Természetesen a fejlesztőket is felsorolhatjuk, ezt a `developers` elem gyermekeivel, a `developer` elemekkel tehetjük meg. Az ügyfelekkel való kapcsolattartás céljából egy levelezőlistát is megadhatunk a `mailingLists` elem segítségével. A projektünkhöz kapcsolódó tárolókat a `reporsitories` elemen belül sorolhatjuk fel.

A POM által nyújtott eszközökről részletesebben a [3]-ban olvashatunk.

3.2. Maven összefoglalás

Ebben a fejezetben az olvasó betekintést nyerhetett a Maven működésébe és a Maven projektek lelkébe, a POM felépítésébe. Bemutattam, milyen eszközökkel lehet leírni egy Maven által kezelt Java projektet. Ezen tudásra építve a következő fejezetben ismertetem az Apache Maven bővítményfejlesztés alapjait.

4. fejezet

Maven plugin

A Maven plugin egy vagy több egymással kapcsolatban lévő úgynevezett MOJO-ból[1] áll. Egy Maven bővítmény projekt tehát nem más, mint Java osztályok és a projektet leíró POM állomány együttese. Hogy kódunk futtatható Maven plugin legyen, a bővítmény fő osztályában meg kell adni egy `@goal` *cél* Javadoc annotációt, ahol a *cél* egy tetszőleges Java azonosító. Az ilyen Javadoc annotációval rendelkező osztályokat a plugin konfigurációs állományában is fel kell tüntetni. Példa egyszerű MOJO-ra:

```
package simple.plugin;

import org.apache.maven.plugin.AbstractMojo;
import org.apache.maven.plugin.MojoExecutionException;

/**
 * Kiírja a "Helló  világ!" szöveget.
 * @goal sayhi
 */
public class SimpleMOJO extends AbstractMojo {
    public void execute() throws MojoExecutionException {
        getLog().info("Helló világ!");
    }
}
```

MOJO-nknak ezen kívül az `org.apache.maven.plugin.AbstractMojo` absztrakt osztályt is ki kell terjeszteni, amely biztosítja a MOJO implementálásához szükséges infrastruktúrát, kivéve az `execute` metódust, melynek implementálása a mi dolgunk. A példában látható kötelező Javadoc annotáció mellett számos Javadoc annotáció áll rendelkezésünkre, amivel beállíthatjuk, hogyan és mikor fusson MOJO-nk, ezekből néhányat később ismertetek.

[1] Maven plain Old Java Object

Az `execute` metódus két kivételt dobhat, hogy informálja az őt futtató Maven-t az esetlegesen bekövetkező hibákról:

- `org.apache.maven.plugin.MojoExecutionException`-t, ha a MOJO nem várt problémába ütközik. E kivétel eldobása esetén a `BUILD ERROR` üzenetet kapjuk.
- `org.apache.maven.plugin.MojoFailureException`-t, ha várt probléma történik (mint például fordítási hiba). Eldobás esetén szintén `BUILD ERROR` üzenetet kapunk.

Az AbstractMojo-ban definiált `getLog` metódus egy naplózó objektummal tér vissza, ami lehetővé teszi, hogy a plugin `debug`, `info`, `warn`, és `error` szintű üzeneteket küldhessen a felhasználónak.

A plugin elkészítéséhez a MOJO megírása után már csak annyi a teendő, hogy a projektleíróban – a POM-ban – elvégezzük a megfelelő beállításokat. A `groupId`-t, az `artifactId`-t és a `version`-t a már ismertetett módon kell megadni. Emellett be kell állítani a `maven-plugin` értéket a `packaging` elemre, és a `dependencies` elemnek legalább egy függőséget meg kell adni, ami a *Maven Plugin Tools API*-ra vonatkozik, hogy feloldjuk az `AbstractMojo`-t és a hozzá kapcsolódó osztályokat. Példa `SimpleMOJO` osztályom leírójára:

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>simple.plugin</groupId>
    <artifactId>hello-maven-plugin</artifactId>
    <packaging>maven-plugin</packaging>
    <version>1.0</version>
    <name>Egyszerű Helló Világ Maven Plugin</name>
    <dependencies>
        <dependency>
            <groupId>org.apache.maven</groupId>
            <artifactId>maven-plugin-api</artifactId>
            <version>2.0</version>
        </dependency>
    </dependencies>
</project>
```

Kész Maven bővítményünk használatához a következő parancsokat kell kiadnunk:

- `mvn compile` – lefordítja a plugin Java kódját
- `mvn test` – futtatja a plugin egységtesztjeit

- `mvn package` – felépíti a plugin JAR-t
- `mvn install` – a plugin JAR-t a **helyi** tárolóba telepíti
- `mvn deploy` – a plugin JAR-t a **távoli** tárolóba telepíti

4.1. MOJO futtatása

A futtatás egyik módja, hogy egy projekt POM-jában beállítjuk a következőket:

```
...
    <build>
        <plugins>
            <plugin>
                <groupId>simple.plugin</groupId>
                <artifactId>hello-maven-plugin</artifactId>
                <version>1.0</version>
            </plugin>
        </plugins>
    </build>
...
```

majd a parancssorban megadjuk a plugin nevét és a célt a következő formában:

```
mvn simple.plugin:hello-maven-plugin:1.0:sayhi
```

ahol a `sayhi` a cél. A verziószám elhagyható, ha a legutolsó verziót szeretnénk futtatni. Ha a *név*`-maven-plugin` vagy `maven-`*név*`-plugin` konvenciót alkalmazzuk projektünk elnevezésére, ahol a *név* tetszőleges, szintén lerövidíthetjük a futtatáshoz szükséges parancsot, a példa parancs a következő lesz:

```
mvn hello:sayhi
```

Megjegyezném, hogy a `maven-`*név*`-plugin` konvenciót azokra a bővítményekre használják, amelyek az Apache Maven projekt részei. Arra is van mód, hogy a konvenció nem betartásakor is rövidüljön a parancs, ilyenkor a `$user.home/.m2/settings.xml` fájlba a következőket kell hozzáadni:

```
<pluginGroups>
    <pluginGroup>simple.plugin</pluginGroup>
</pluginGroups>
```

Pluginunkat hozzárendelhetjük egy adott projekt build életciklusának bizonyos fázisához, ha az adott projekt POM-jában megadjuk pluginunk célját például a következő módon:

```
...
    <build>
        <plugins>
            <plugin>
                <groupId>simple.plugin</groupId>
                <artifactId>hello-maven-plugin</artifactId>
                <version>1.0</version>
                <executions>
                    <execution>
                        <phase>compile</phase>
                        <goals>
                            <goal>sayhi</goal>
                        </goals>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
...
```

Ezen beállítás esetén a plugin cél lefut a Java kód minden egyes fordításakor.

4.2. Paraméterek

A paraméterek fontos szerepet játszanak a MOJO-k életében, hiszen paraméter segítségével érhetjük el egy adott projekt POM-ját, ezáltal azok elemeit és az elemek értékeit, így a MOJO műveletvégzésére nézve is fontos beállításokat végezhetünk el.

Paramétereket úgy definiálhatunk, hogy létrehozunk egy példányváltozót, és hozzáadjuk a megfelelő Javadoc annotációt. Példa egy egyszerű MOJO paraméterére:

```
/**
 * Az üdvözlő üzenet.
 *
 * @parameter expression="${sayhi.greeting}"
 *     default-value="Helló Világ!"
 */
private String greeting;
```

A `@parameter` Javadoc annotáció segítségével nevezhetünk ki egy változót MOJO paraméterré, aminek `default-value` paraméterével definiálhatjuk a változó alapértelmezett értékét. Ez az érték projektre hivatkozó kifejezéseket is tartalmazhat, mint például

`$projekt.version`. A kifejezés paraméter használatával konfigurálhatjuk MOJO paraméterünket a parancssorból, egy rendszertulajdonságra hivatkozással, amit a felhasználó a `-D` kapcsolón keresztül tehet meg.

Egy plugin paraméter értékének beállítását Maven 2 és Maven 3 projektben az adott projekt leírójában végezhetjük el a pluginok definiálásának részeként. Egy példa plugin konfigurálására:

```
...
    <plugin>
        <groupId>simple.plugin</groupId>
        <artifactId>hello-maven-plugin</artifactId>
        <version>1.0</version>
        <configuration>
            <greeting>Szervusz</greeting>
        </configuration>
    </plugin>
...
```

A `configuration` részben a `greeting` elem a paraméternév, aminek `Szervusz` tartalma az érték, amit a paraméterhez rendelünk. A továbbiakban bemutatom, milyen típusú paramétereket kezel egy Maven plugin.

4.2.1. Egyszerű paramétertípusok

MOJO egyszerű típusú paramétereihez használhatjuk a Java által ismert primitív típusokat (mint például `int`, `boolean`, `char`) és azok csomagolóosztályait (`Integer`, `Boolean`, `Character`, stb.), továbbá a Java `Date`, `File`, `URL`, `String` és `StringBuffer` osztályait. Példa `Integer` típusú paraméter értékadására POM-ból:

```
/**
 * Egy Integer paraméter.
 *
 * @parameter
 */
private Integer anInteger;
```

Konfiguráció:

```
<anInteger>32</anInteger>
```

A `Date` típusú paraméter értékadásakor a POM-ban megadott karakterlánc dátum típusúvá konvertálódik a `DateFormat.parse()` metódus alkalmazásával. Példa erre:

```
/**
 * Ezen példa megalkotásának időpontja.
 *
 * @parameter
 */
private Date exampleDate;
```

Konfiguráció:

```
<exampleDate>2011-04-05 5:46:27.3 PM</exampleDate>
```

4.2.2. Összetett paramétertípusok

MOJO összetett típusú paraméterének használhatunk tömböt, Java kollekciókat (vagyis minden olyan osztályt, ami implementálja a `java.util.Collection` interfészt), olyan map osztályt, ami implementálja a `java.util.Map` interfészt, mint például a `HashMap`, de nem terjeszti ki a `java.util.Properties` osztályt, és olyan tulajdonság osztályt, ami kiterjeszti a `java.util.Properties`-t. Ez is egyfajta map, de ezt a MOJO külön kezeli. Ráadásként olyan egyéb osztályt is adhatunk meg MOJO paraméter típusának, ami nem implementálja a `java.util.Map` és `java.util.Collection` interfészt, és nem terjeszti ki a `java.util.Dictionary` osztályt. Példa tömb használatára:

```
/**
 * Karaktertömb.
 *
 * @parameter
 */
private String[] stringArray;
```

Értékadás a POM segítségével:

```
<stringArray>
    <param>Wincs Eszter</param>
    <param>Gipsz Jakab</param>
</stringArray>
```

A listák értékadása analóg módon történik a tömbök értékadásával, Map-eknél a `param` elem helyett `key1`, `key2`, stb. elemeket kell megadni.

A paramétertípusok részletesebb leírásáról és MOJO-beli beállító metódusok használatáról a *Maven – Guide to Developing Java Plugins* [2] honlapján olvashatunk.

4.3. Maven plugin összefoglalás

A fejezetben bemutattam, hogy mik a Maven bővítmények szerkezeti követelményei, hogyan kell felépíteni egy MOJO-t és a bővítmény POM-ját. Röviden ismertettem, hogyan tudunk paramétereket definiálni pluginokhoz és ezek értékeit hogyan állíthatjuk be. Ezzel el is érkeztem szakdolgozatom tárgyához, az általam fejlesztett DOAP generáló bővítményhez, amit a következő fejezetben részletesen leírok.

5. fejezet

Bővítményem

Most, hogy a szakdolgozatomhoz felhasznált technológiákat ismertettem, rátérnék az általam fejlesztett Apache Maven bővítményre, a *zsolczai-doap-maven-plugin*ra. Pluginom lényegi része mindössze egy Java osztály és egy XSLT stíluslap, ezen eszközökkel valósítom meg a DOAP generálást egy adott projekt POM-jából.

5.1. zsolczai.doap.DoapGenerator Java osztály

```
/**
 * A Project Object Management fájl felhasználásával egy DOAP
 * (Description of a Project) leírást készít a projektről.
 *
 * @goal create
 */
public class DoapGenerator extends AbstractMojo {
    ...
```

MOJO osztályom egy cél (`@goal`) Javadoc annotációval rendelkezik, melynek neve `create`. A plugin telepítése után parancsként a plugin nevét, és ezt a célt kell megadni, hogy futtathassuk adott Java projektünkön a már ismertetett alakban:

```
mvn zsolczai-doap:create
```

```
    ...
    /**
     * A POM fájl transzformációját elősegítő stíluslap.
     */
    private static final String XSL_STYLESHEET =
            "default.xsl";
```

```
/**
 * A DOAP fájl neve.
 */
private static final String DOAP = "doap.rdf";

/**
 * A Project Object Manage fájl.
 * @parameter default-value="pom.xml";
 */
private File pom;
...
```

A három MOJO attribútum közül két osztályszintűben tárolom rendre az XSLT stíluslap és a generálandó DOAP állomány nevét. A harmadik attribútum egy MOJO paraméter, pluginom ezen keresztül fér hozzá annak a Maven projektnek a POM állományához, amelyhez DOAP leírást szeretnénk készíteni.

```
...
public void execute() throws MojoExecutionException {
    try {
        File targetDIR = new File("target");
        File siteDIR = new File("target/site");

        if (!targetDIR.exists()) {
            siteDIR.mkdirs();
        } else if (!siteDIR.exists()) {
            siteDIR.mkdir();
        }
        ...
```

MOJO-m a belépési pontjában az `execute()` metódusban első lépésként ellenőrzi, hogy a cél Maven projekt rendelkezik-e a szükséges könyvtárszerkezettel a DOAP állomány tárolásához. Ha nem, akkor ez azt jelenti, hogy bővítményem futtatása előtt a projekten nem futtattak weboldal (site) generáló plugint, ebben az esetben elkészíti a hiányzó könyvtárakat (a projekt gyökérkönyvtárában a `target`, azon belül a `site` mappát).

```
        ...
        getTransformer().transform(new StreamSource(pom),
                new StreamResult("target/site/" + DOAP));
        linkDoapToHtml();
    } catch(TransformerException e) {
        throw new MojoExecutionException(
                "XML transzformációs Hiba!", e);
    } catch(IOException e) {
```

```
        }
    }
    ...
```

Második lépésként a `getTransformer()` metódus által átadott transzformációs objektumnak (jelen esetben az XML transzformációs stíluslapnak) átadom forrás csatornaként a cél projekt POM állományát, és cél csatornaként az elkészítendő DOAP állományt. A DOAP generálásáért ezután a `javax.xml.transform.Transformer` példány gondoskodik. Ahhoz, hogy hogyan is kapom meg a kívánt transzformációs objektumot, vagyis a `getTransformer()` metódus implementációját később mutatom be.

A `linkDoapToHtml()` metódus megkísérli a cél projekt `index.html` oldalához linkelni a DOAP állományt metaadatként, már ha a projekthez rendelkezik a *site* bővítmény segítségével elkészített API specifikációs weboldallal. A `javax.xml.transform.TransformerException` kivétel, amit pluginom egy `org.apache.maven.plugin.MojoExecutionException` kivételként dob tovább, abban az esetben következik be, ha az XSLT stíluslap hibás, vagy nem létezik. `java.io.IOException` kivételt a `linkDoapToHtml()` metódus dobhat, ezt a kivételt MOJO-m csak elkapja, és nem kezeli, mert ez abban az esetben váltódik ki, ha az `index.html` állomány nem létezik, és ilyenkor elég, ha egyszerűen nem hajtódik végre a DOAP metaadatként való linkelése a weboldalhoz, nem szükséges, hogy a felhasználót informálja az állomány hiányáról.

```
    ...
    private static Transformer getTransformer()
            throws TransformerException {
        TransformerFactory factory =
                TransformerFactory.newInstance();
        InputStream in = DoapGenerator.class
                .getResourceAsStream("/" + XSL_STYLESHEET);

        if (in != null) {
            return factory.newTransformer(new StreamSource(
                    in));
        }
        return factory.newTransformer();
    }
    ...
```

Az előbbi kódrészletben meghívott `getTransformer()` metódus a `javax.xml.transform.TransformerFactory` objektum gyár osztály segítségével készíti el a transzformációs objektumot. Ha a MOJO megtalálta a

`getResourceAsStream()` metódus paramétereként megadott helyen lévő XSLT stíluslapot, akkor a transzformációs objektum e stíluslap segítségével készíti el a cél projekt POM állományából a nyert információk alapján a DOAP fájlt, ha nem találta, vagyis az `in` változónak null referencia adódott át, abban az esetben metódus egy üres transzformációs objektumot ad át hívójának.

```
    ...
    private void linkDoapToHtml() throws IOException {
        BufferedReader in = new BufferedReader(
                new FileReader("target/site/index.html"));
        StringBuilder index = new StringBuilder();
        String line = in.readLine();
        boolean linked = false;

        while (line != null) {
            if (!linked && line.contains("</head")) {
                index.append("<link rel=\"meta\"")
                        .append(title=\"DOAP\" ")
                        .append("type=\"application/rdf+xml")
                        .append(\" href=\"").append(DOAP)
                        .append("\"/>").append("\n");
                linked = true;
            }
            index.append(line).append("\n");
            line = in.readLine();
        }
        in.close();
        FileWriter out =
                new FileWriter("target/site/index.html");
        out.write(index.toString());
        out.close();
    }
}
```

`linkDoapToHtml()` metódusom úgy valósítja meg a generált DOAP cél projekt weboldalához metaadatként linkelését, hogy egy pufferelt olvasóegység (`java.io.BufferedReader`) példány segítségével addig olvassa soronként az `index.html` tartalmát egy `java.lang.String` példányba, amíg el nem éri a `</head>` zárócímkét, vagyis, amíg a weboldal fejlécének végéhez nem ér. Közben a ciklusban a sztring tartalma átíródik egy `java.lang.StringBuilder` példánynak, a metódus így őrzi meg a HTML állomány tartalmát. Amint elér a fejléc végéhez, a `</head>` zárócímke elé beilleszti a hivatkozást a StringBuilder példányba, majd folytatódik a további sorok bemásolása. A HTML

fájl tartalmának kiolvasása után a metódus újra megnyitja az állományt, de ezúttal felülírja azt, annak módosított tartalmával.

MOJO osztályom után most következzen az XSLT transzformációs lap.

5.2. default.xsl XSLT stípuslap

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE xsl:stylesheet [
    <!ENTITY lang "en">
    <!ENTITY rdf
            "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <!ENTITY doap "http://usefulinc.com/ns/doap#">
    <!ENTITY foaf "http://xmlns.com/foaf/0.1/">
]>
...
```

Stíluslapomhoz entitásokat deklaráltam a készítendő DOAP szótár által használt névterek és a leírás nyelvének kiemelésére.

```
...
<xsl:stylesheet
        xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
        xmlns:pom="http://maven.apache.org/POM/4.0.0"
        version="1.0">

    <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
    ...
```

A stíluslap gyökérelemében deklarálom az `xsl` és a `pom` névteret. Az `xsl:output` elemben megadom a kimenet formátumát, az eredmény dokumentum kódolását, és azt, hogy legyen automatikus szövegtördelés az emberi olvashatóság érdekében.

```
    ...
    <xsl:template match="pom:project">
        <xsl:variable name="proj.homepage" select="pom:url"/>
        <Project xmlns:rdf="&rdf;" xmlns:foaf="&foaf;"
                xmlns="&doap;">
            <xsl:if test="pom:name">
                <name>
                    <xsl:value-of select="pom:name"/>
                </name>
            </xsl:if>
            ...
```

Ha a `pom:project` mintára illeszkedik a forrás POM egy csomópontja, vagyis van benne `project` elem, akkor az `xsl:template` gyermek elemeiként megadott szekvenciával folytatódik a program végrehajtása. A `proj.homepage` változóhoz hozzárendelem a `pom:url` értékét, ha létezik, erre a mintára való illeszkedést később vizsgálom. Létrehozom a `Project` elemet, ami a DOAP leírás gyökéreleme lesz, és hozzárendelem az `rdf`, `foaf`, és `doap` névtér előtagokhoz a megfelelő URI-kat. Ha a stíluslap a POM-ban talál `name` elemet, akkor annak értékét, vagyis a forrás projekt nevét átmásolja a DOAP-ba.

```
...
<xsl:if test="pom:url">
    <homepage rdf:resource="{$proj.homepage}"/>
</xsl:if>

<xsl:if test="pom:description">
    <description xml:lang="&lang;">
        <xsl:value-of select="pom:description"/>
    </description>
</xsl:if>
...
```

Ez a kódrészlet a `pom:url` mintára illeszkedést teszteli, vagyis azt, hogy a POM-ban definiált-e az `url` elem. Ha van, létrehozza a `homepage` elemet, aminek `rdf:resource` attribútumának értékeként átadja a `$proj.homepage` változó értékét. Ezután azt vizsgálja, hogy a POM-ban létezik-e `description` elem, ha igen ezt az elemet létrehozza a DOAP-ban is, ellátja a `lang` entitásban tárolt értékkel, és a POM `description` értékét átadja a DOAP `description` elemnek.

```
...
<xsl:for-each
        select="pom:developers/pom:developer">
    <developer>
        <foaf:Person>
            <foaf:name>
                <xsl:value-of select="pom:name"/>
            </foaf:name>
            <xsl:if test="pom:email">
                <xsl:variable name="mail"
                        select="pom:email"/>
                <foaf:mbox
        rdf:resource="mailto:{$mail}"/>
            </xsl:if>
            <xsl:if test="pom:url">
                <xsl:variable name="homepage"
```

```
                                    select="pom:url"/>
                            <foaf:homepage
                                rdf:resource="{$homepage}"/>
                        </xsl:if>
                        <xsl:if test="pom:organization">
                            <xsl:variable name="org"
                                select="pom:organization"/>
                            <foaf:Organization
                                rdf:resource="{$org}"/>
                        </xsl:if>
                    </foaf:Person>
                </developer>
            </xsl:for-each>
            ...
```

A `developer` elemek vizsgálatára az `xsl:for-each` ciklust választottam, hogy a forrás projekt minden egyes fejlesztője fel legyen tüntetve a DOAP szótárban. A fejlesztőket a FOAF `Person` osztálya segítségével teszi közzé a generált DOAP, `foaf:name` (név), `foaf:mbox` (email cím), `foaf:homepage` (saját weboldal) és `foaf:organization` (a személyt foglalkoztató szervezet) elemekkel, feltéve, hogy azok meg vannak adva a POM állományban.

```
            ...
            <xsl:for-each select="pom:licenses/pom:license">
                <xsl:variable name="proj.license"
                        select="pom:url"/>
                <license rdf:resource="{$proj.license}"/>
            </xsl:for-each>

        </Project>
    </xsl:template>

</xsl:stylesheet>
```

Végül XSLT stíluslapom a `license` elemek létezését vizsgálja, szintén `xsl:for-each` ciklussal, ha a forrás POM-ban esetleg több licensz is hozzá lenne rendelve a projekthez. Ez esetben minden licenszet átvesz a DOAP leírás.

5.3. Példa DOAP kimenetre

Bővítményem kimenetére bemutatok egy példát is, mely a `maven-hello` projektet írja le:

```
<?xml version="1.0" encoding="UTF-8"?>

<Project xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:pom="http://maven.apache.org/POM/4.0.0"
        xmlns="http://usefulinc.com/ns/doap#">
    <name>maven-hello</name>
    <description xml:lang=
        "en">A simple hello-world project.</description>
    <developer>
        <foaf:Person>
        <foaf:name>Gergo Zsolczai</foaf:name>
        <foaf:mbox
            rdf:resource="mailto:zsolczai.gergo@gmail.com"/>
        <foaf:Organization rdf:resource=
            "University of Debrecen, Faculty of Informatics"/>
    </foaf:Person>
    </developer>
    <developer>
        <foaf:Person>
        <foaf:name>En</foaf:name>
        <foaf:homepage rdf:resource="http://www.valaki.hu"/>
        </foaf:Person>
    </developer>
    <license
        rdf:resource="http://www.gnu.org/copyleft/gpl.html"/>
    <license rdf:resource="http://www.fiktivlicensz.hu"/>
</Project>
```

A következő példa a `maven-hello` projekthez generált HTML oldal fejlécének forrását mutatja be, ahova bővítményem beilleszt egy linket a projekt DOAP leírójáról közvetlen a `head` elem záró címkéje elé:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<!-- Generated by Apache Maven Doxia at Apr 22, 2011 -->
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
    lang="en">
    <head>
        <meta http-equiv="Content-Type"
            content="text/html; charset=UTF-8" />
        <title>maven-hello - About</title>
        <style type="text/css" media="all">
            @import url("./css/maven-base.css");
```

```
            @import url("./css/maven-theme.css");
            @import url("./css/site.css");
        </style>
        <link rel="stylesheet" href="./css/print.css"
            type="text/css" media="print" />
        <meta name="Date-Revision-yyyymmdd"
            content="20110422" />
        <meta http-equiv="Content-Language" content="en" />

        <link rel="meta" title="DOAP"
            type="application/rdf+xml" href="doap.rdf"/>
    </head>
    ...
```

Végezetül bemutatnám a fentebbi `maven-hello` DOAP leírásomat HTML oldalként megjelenítve. Ezt a megjelenítést az *OpenLink Data Explorer* nevű szoftver segítségével lehet megvalósítani, mely kiegészítőként elérhető a Mozilla Firefox, a Google Chrome és az Apple Safari webböngészőkhöz. Ezen kiegészítő lehetővé teszi, hogy egy weboldal nyers adatait és a köztük lévő rejtett kapcsolatokat böngészhessük. Az OpenLink Data Explorer a szemantikus web egyfajta megvalósításán, a *Linked Data*-n alapszik. További részleteket arról, hogyan készítsük el weboldalunkat a Linked Data szellemében, Tim Berners-Lee *Design Issues: Linked Data* cikkében [4] olvashatunk.

Ezzel el is értem Apache Maven bővítményem bemutatásának végéhez, ami a felsorolt eszközökkel valósítja meg a DOAP leírás készítését egy projekt POM fájljának felhasználásával.


OPENLINK SOFTWARE Data Explorer
What Where When Who Images Grid view Tag Cloud SVG Graph Navigator Custom
This view shows all RDF data grouped by subject resource.
Cache All: Check Uncheck Invert Sel Purge Refresh Permalink
Total 16 triples
file:///tarolo/Dokumentumok/Sz...ven-hello/target/site/doap.rdf - 16 triples - Remove - Refresh - Permalink
Order By Resource Category
rdf:type, namespacelang, foaf:Person, foaf:name, foaf:mbox, foaf:Organization, foaf:homepage, rdf:resource
8 records (16 triples, 16 properties) match selected filters.
To enable grouping, order records by some value.
_:480_0
rdf:type doap:name
_:480_1
rdf:type doap:description
namespacelang en
_:480_2
rdf:type doap:developer
foaf:Person _:480_3
Gergo Zsolczai
foaf:name Gergo Zsolczai
foaf:mbox zsolczai.gergo@gmail.com
foaf:Organization University of Debrecen, Faculty of Informatics
_:480_4
rdf:type doap:developer
foaf:Person _:480_5
En
foaf:name En
foaf:homepage http://www.valaki.hu
_:480_6
rdf:type doap:license
rdf:resource http://www.gnu.org/copyleft/gpl.html
_:480_7
rdf:type doap:license
rdf:resource http://www.fiktivlicensz.hu
POWERED BY VIRTUOSO LINKINGOPENDATA
About OpenLink Data Explorer
Copyright © 2005-2011 OpenLink Software

6. fejezet

Összefoglalás

Ezzel el is érkeztünk szakdolgozatom záró fejezetéhez, melyben röviden összefoglalom munkám eredményét.

Az első fejezetben ismertettem a *Szemantikus világháló* jelentését, mi rejlik ezen elgondolás mögött. Megvalósításával lehetőség nyílna a webes tartalmak struktúrálására az erőforrásokhoz rendelt metainformációk és a köztük lévő kapcsolatok jelölésére, és sémák segítségével az adatok alapján következtetések levonására. Ezzel hatékonyabban tudnánk a világhálón keresni, nagyobb valószínűséggel találnánk meg a nekünk megfelelő weboldalt, képet, és lényegében bármit, amire a web mai formájában oly nehezen bukkanunk rá, ha egyáltalán sikerrel jár keresésünk.

Röviden bemutattam az *XML* nyelvet, mely sikerével bizonyítja, mennyire alkalmas különböző platformmal rendelkező gépek közötti adatcserére, ember általi olvashatósága pedig elősegíti a dokumentumban esetlegesen felmerülő hibák könnyű javítását. Kitértem arra is, hogyan lehet biztosítani *XML sémák*kal, hogy az XML alapú adatcserét bonyolítók között ne legyen félreértés az adatok jelentését illetően. Emellett kitértem a bővítményem által felhasznált egyik technológiára, az *XSLT transzformációs stíluslap*ra is, mellyel XML dokumentumok transzformálását lehet elvégezni más XML dokumetumokká.

Ezután ismertettem az *RDF* keretrendszert, melynek XML szintaxissal rendelkező formája megfelelő alapként szolgál a szemantikus világháló megvalósítására, az erőforrásokhoz metaadatok rendelésére. Bemutattam, hogy *RDF sémák* segítségével megvalósíthatjuk a szemantikus web másik alapgondolatát, metaadatok alapján következtetések levonását. Felhívtam a figyelmet arra, hogy amellett, hogy az RDF séma és az objektum-orientált világ között néhány hasonlóság figyelhető meg, a különbség az, ami a webes tartalmak leírására alkalmasabbá teszi az OO nyelvekkel szemben, ez pedig a tulajdonságközpontúság. Ez biztosítja a rugalmasságot, hogy az állandóan változó világhálót hatékonyan tartsuk karban.

A *DOAP* bemutatásával lefedtem azokat a szemantikus webhez kapcsolódó technológiákat, melyeket felhasználtam szakdolgozatomban. Az adott fejezetben leírtam, hogy DOAP segítségével egy RDF-re épülő XML szintaxisú szótárat készíthetünk, mellyel – mint ahogy azt neve is mutatja: Description of a Project (egy Projekt Leírása) – szoftver projektek adatait tehetjük közzé.

Mivel szakdolgozatom témája az Apache Maven bővítmény fejlesztés, ezért a Maven projekt kezelő rendszer ismertetése sem maradhatott ki, ezért a Maven fejezetben röviden bemutattam ezen projekt kezelő rendszerben rejlő lehetőségeket és azokat az eszközöket, amelyeket projektünk implementációs és tesztelési fázisát kezelhetjük.

A Maven plugin fejezetben pedig azokat az eszközöket ismertettem, amelyeket a Maven bővítményei fejlesztéséhez biztosít. Ahogy azt már említettem a felkínált eszközökhöz a Maven részletes leírást is ad *Maven – Guide to Developing Java Plugins* weboldalán[2], így ehhez a fejezethez én is felhasználtam forrásként.

A Bővítményem című fejezetben pluginom forráskódrészletei bemutatása után leírtam, hogyan és milyen műveletet hajt végre az adott rész és az XML feldolgozóm milyen parancsokat kap a bővítményemhez készült XSLT stíluslaptól.

7. fejezet

Köszönetnyilvánítás

Köszönetet szeretnék nyilvánítani Jeszenszky Péternek, aki témavezetőmként segítőkészségével nagyban hozzájárult szakdolgozatom elkészítéséhez, gyakorlatvezető tanárként pedig tanulmányaim gyakorlati elmélyítéséhez.

Továbbá megköszönném a Debreceni Egyetem mindazon oktatójának munkáját, akik kiváló szakmai tudásukkal hozzájárultak egyetemi szintű matematikai és informatikai tudásom megszerzésére, kiemelve Dr. Juhász István tanár urat, aki sajátos előadásmódjával talán a legnagyobb ösztönzést adta, hogy szakmai tudásom gyarapítsam a programozás területén. Már több hallgatótársamtól hallottam és én is csak egyet érthetek velük, hogy amit ő tanított, arra emlékszem a legjobban, az maradt meg bennem a legrészletesebben.

Irodalomjegyzék

[1] FOAF Vocabulary Specification. URL http://xmlns.com/foaf/spec/.

[2] Maven – Guide to Developing Java Plugins. URL http://maven.apache.org/guides/plugin/guide-java-plugin-development.html.

[3] POM Reference. URL http://maven.apache.org/pom.html.

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

[5] Brian McBride Dan Brickley, R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, 2004. URL http://www.w3.org/TR/rdf-schema/.

[6] Brian McBride Dave Beckett. RDF/XML Syntax Specification, 2004. URL http://www.w3.org/TR/rdf-syntax-grammar/.

[7] Edd Dumbill. XML Watch: Describe open source projects with XML, Part 1, 2004. URL http://www.ibm.com/developerworks/xml/library/x-osproj.html.

[8] Edd Dumbill. XML Watch: Describe open source projects with XML, Part 2, 2004. URL http://www.ibm.com/developerworks/xml/library/x-osproj2/.

[9] Edd Dumbill. XML Watch: Describe open source projects with XML, Part 3, 2004. URL http://www.ibm.com/developerworks/xml/library/x-osproj3/.

[10] Murray Maloney Noah Mendelsohn Henry S. Thompson, David Beech. XML Schema Part 1: Structures, 2004. URL http://www.w3.org/TR/xmlschema-1/.

[11] Ashok Malhotra Paul V. Biron, Kaiser Permanente. XML Schema Part 2: Datatypes, 2004. URL http://www.w3.org/TR/xmlschema-2/.

[12] Jeszenszky Péter. Resource Description Framework. URL https://www.inf.unideb.hu/~jeszy/download/semweb/RDF.pdf.

[13] Benkő Tamás Szerendi Péter, Lukácsy Gergely. *A szemantikus világháló elmélete és gyakorlata*. Typotex Elektronikus Kiadó Kft., 2005. ISBN 978-963-9548-48-0.