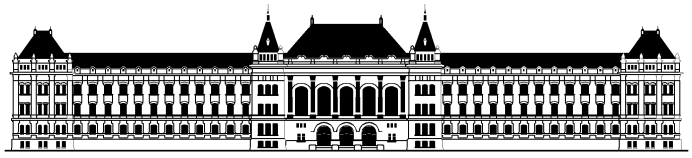


BUDAPESTI MŰSZAKI ÉS GAZDASÁGTUDOMÁNYI EGYETEM

VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR

IRÁNYÍTÁSTECHNIKA ÉS INFORMATIKA TANSZÉK

A WIKIPÉDIA VALÓSIDEJŰ NYELVI FELDOLGOZÁSA OSGI ALAPOKON

SZAKDOLGOZAT

Készítette
UNICSOVICS MILÁN GYÖRGY

Konzulens
HÉDER MIHÁLY, SIMON BALÁZS

2013. november 7.

Tartalomjegyzék

HALLGATÓI NYILATKOZAT

Alulírott *Unicsovics Milán György*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2013. november 7.

Unicsovics Milán György
hallgató

Kivonat

magyar

Abstract

english

Bevezető

Mai világunkban a tudás a legnagyobb érték. A tudás valójában kontextusba ágyazott információ, mely elemi adatokból épül fel. Hosszú ideje irányulnak kutatások és fejlesztések az informatikában a tudás minél hatékonyabb megszerzésére, azonban, ha egy nehéz és összetett problémát akarunk megoldani valamilyen módszerrel, azt tapasztaljuk, hogy a hozzá szükséges tudás megszerzése lesz mindig a szűk keresztmetszet (ez az ún. *knowledge acquisition bottleneck*).

Ez a helyzet egy ideje a különféle adatábrázolási módszereknek köszönhetően kezdett megváltozni. A struktúrálatlan nehezen feldolgozható adatok mellett megjelentek a struktúrált és szemi-struktúrált adatok, melyekből az információt sokkal gyorsabban és könnyebben lehet kinyerni, és olyan automatizált folyamatokban is felhasználhatóak már, ahol korábban emberi közreműködés lett volna szükséges. Megjelentek a kollaboratívan szerkeszthető tudásbázisok, ezek közül is a legnépszerűbb és legnagyobb a Wikipedia, mely egy szemi-struktúrált információforrás. Ezen tudásbázisokat főleg a mesterséges intelligencia [**?**], természetes nyelvi feldolgozás, valamint a számítógépes nyelvészet területén, de az informatika szinte minden ágában ugyanúgy felhasználják.

A szemi-struktúrált információforrások, mint a Wikipedia (továbbiak például a Flickr, Twitter vagy a Yahoo! Answers) egyesítik a struktúrált és struktúrálatlan források előnyeit, így rendkívül jó kiindulópontjai különféle kutatásoknak. A struktúrálatlan adatokkal szembeni előnye, hogy gépek számára is értelmezhető információt tárolnak; a struktúrált adatok előállításánál és kezelésénél pedig kevésbé erőforrás-igényes a szemi-struktúrált adatok létrehozása és karbantartása.

Az informatika ezen területén történő kutatások már 2005-2006 körül megkezdődtek, azonban az ezek eredményeit felhasználó fejlesztések száma nem túl sok. Célom tehát az eddig elkészült és elérhető fejlesztések vizsgálata, információk és tapasztalatok összesítése, valamint ezeket továbbfejlesztve egy korszerűbb rendszer összeállítása, melyre alapozva eredményes kutatásokat lehet kezdeményezni, a legnagyobb elérhető szemi-struktúrált erőforrás a Wikipedia segítségével, a lehető legflexibilisebb módon.

1. fejezet

Hasonló megoldások vizsgálata

Mielőtt a saját Wikipedia alapú rendszer tervezésének nekiláttam volna megvizsgáltam a hasonló célú, elérhető megoldásokat, hogy utána a tapasztalatokból kiindulva láthassak neki a munkának.

WikiNet[?] A WikiNet egy teljesen struktúrált adatszerkezetű tudásbázist, másnéven ontológiát épít fel. Ez egy Perl nyelvű szkriptek gyűjteményéből álló megoldás, a Wikipedia egy statikus, letöltött verzióját használja forrásként, a kinyert fogalmakat és kapcsolatokat külön szöveges fájlokban tárolja el.

Elérhetőség: `http://www.h-its.org/english/research/nlp/download/wikinet.php`

DBpedia Ez a megoldás a Wikipedia dump-ján alapul, belőle tényszerű információkat nyer ki és rögzít struktúrált formában, melyet ezután a weben közzétéve a felhasználók számára egy SQL szerű nyelvvel lekérdezhetővé tesz. A alkalmazás Scala, Java nyelven íródott, adatbázisként Virtuoso Universal Server-t használ.

Elérhetőség: `http://dbpedia.org/`

BabelNet[?] A BabelNet egy Java nyelven írt alkalmazás, ontológiát készít más adatforrások (Wikipedia dump, WordNet) alapján és ezekből egy ”enciklopédikus szótárat” készít a felhasználók számára. Egy adott fogalomra keresve a szemantikailag kapcsolódó fogalmak is megjelennek, valamint ezek szinonímái (a BabelNet elnevezése szerint *synset*-ek), és minden szinonímához tartozik egy rövid definíció (*gloss*) több nyelven.

Elérhetőség: `http://lcl.uniroma1.it/babelnet/`

Java Wikipedia Library[?] A JWPL egy Java alapú alkalmazás, mely egy interfészt ajánl ki, amivel a Wikipedia tartalmához lehet hozzáférni. A JWPL tartalmaz egy Mediawiki Markup parser-t, mellyel a letöltött Wikipedia dump-ot beolvassák, a Wikitext-ből átalakított szövegekből optimalizált adatbázisokat készítenek, melyekhez végül hozzáférési felületet nyújtanak.

Elérhetőség: `http://www.ukp.tu-darmstadt.de/software/jwpl/`

Wikipedia Preprocessor Ezzel az eszközzel más alkalmazások számára lehet egy Wikipedia dump-ot feldolgozhatóbb formába hozni. A statikus dump-ból kiinduló alkalmazásoknál

erre szükség is van, mivel a például Wikipedia 2007. július 19-i XML formátumban letölthető tartalma is 12 GB méretű, amit nyers formában szinte lehetetlen hatékonyan felhasználni.

Elérhetőség: `http://www.cs.technion.ac.il/~gabr/resources/code/wikiprep/`

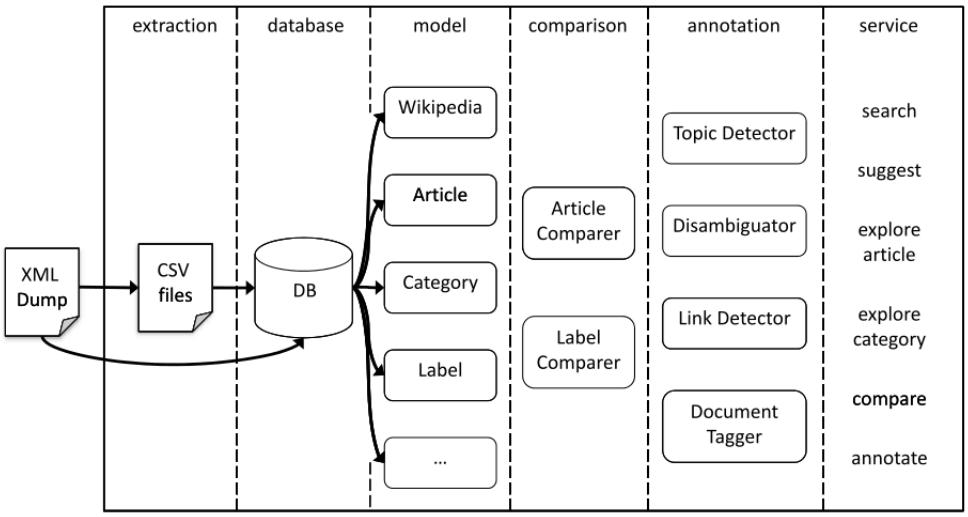
YAGO2[?] A YAGO2 a Wikipedia egy korábban letöltött tartalmán, és egyéb online tudásbázisokon (WordNet, GeoNames) alapuló ontológia. Az ontológiához több formátumban lehet hozzáférni, ezt kisebb Java nyelven írt konvertáló eszközökkel lehet megtenni.

Elérhetőség: `http://www.mpi-inf.mpg.de/yago-naga/yago/`

1.1. Egy ígéretes megoldás: Wikipedia Miner

A Wikipedia Miner egy olyan nyílt forráskódú teljes egészében Java nyelven írt eszközkészlet, mellyel lehetővé válik kutatók és fejlesztők számára, hogy alkalmazásukban egyszerűen hozzáférjenek a Wikipedia tartalmához. A hozzáférési felületet Java API-n keresztül nyújtja az alkalmazás, a tudásbázis tartalmát pedig egy összegzett formában használja fel. Ezenfelül a rendszer olyan technológiákat használ fel, mint az elosztott számítási keretrendszert nyújtó Apache Hadoop és a Weka adatbányászati szoftver.

A Wikipedia Miner alkotói szerint három lehetőség van a szemi-struktúrált adatforrások felhasználására: vagy kész ontológiákat használunk, mint például a YAGO2, vagy egy nyers letöltött dump-ból elkészítjük a saját ontológiánkat, mint például a JWPL és az alapján kezdünk dolgozni; a harmadik lehetőség, hogy folyamatosan frissítjük az adatforrásunkat, így mindig naprakész forrást használunk.



1.1. ábra. Wikipedia Miner architektúra (forrás: Artificial Intelligence, Wikipedia and Semi-Structured Resources[**?**])

A rendszer áttekintő architektúráját mutatja be a 1.1. ábra. A rendszer belépési pontjára érkezik egy nagy méretű XML formátumú fájl, mely a Wikipedia hivatalos API-ján keresztül lett letöltve és tartalmazza a naprakész Wikipedia teljes állományát.

Az *extraction* package funkciója, hogy kinyerjen az XML forrásból egy összegzett tartalmat. Az

adatkinyerési folyamat használja a nagy méretű XML fájl feldolgozásához a Hadoop és MapReduce technológiákat, valamint a Google GFS fájlrendszerét. Az adatok kinyerése tehát elosztott rendszeren történik, a 27 GB méretű teljes angol Wikipedia feldolgozása a mérések szerint egy 2 magos, 2,66 GHz órajelű processzorral és 4 GB memóriával rendelkező 30 gépből álló clusternek 2,5 órájába telik.

A Wikipedia kivonatolása után a teljes XML fájl és az összegzett tartalmak is bekerülnek a *database* package adatbázisába. Adatbázisként Berkeley DB Java Edition-t használnak, mellyel akár egy teljes adatbázist is a memóriában lehet tartani, ezzel nagyon gyors lekérdezhetőséget elérve.

A Wikipedia Miner keretrendszert használók a tartalmakhoz a *model* package absztrakciós felületen keresztül férhetnek hozzá, mely becsomagolja a kinyert információkat és egy jól dokumentált osztályok formájában teszi közzé (mint például: `Wikpedia`, `Article`, `Category`).

A további csomagok (*comparison* és *annotation*) már a felhasználók számára használható szemantikus tartalom kezelését segítő eszközök, a *service* package-ben pedig konkrét szolgáltatások találhatóak, melyeket felhasználhatnak a fejlesztők, illetve kiegészíthetik őket.

1.2. Tanulságok

Amint látható a fentebb leírt Wikipedia-t felhasználó eszközök szinte mind a Wikipedia egy korábban letöltött változatán (dump) alapul, mely módszernek több hátránya is van. A letöltött tudásbázis mérete rendkívül nagy, így azt kezelni nehézkes, feldolgozási ideje rendkívül hosszú (a Wikipedia Processor feldolgozási ideje például körülbelül 43,5 óra). A hosszú feldolgozási idő nem mindig engedhető meg, ráadásul amíg nincs feldolgozott forrás, a rendszer sem működőképes. Újabb dump letöltésekor kezdhető előlről a feldolgozás, így többször is megakaszthatja ez a folyamat a rendszer működését, a rendelkezésre állási időt csökkentve.

A másik megoldás, melyet a Wikipedia Miner (**??**. alfejezet) is demonstrál a Wikipedia folyamatos *on-the-fly* feldolgozása. Míg az előző módszer figyelmen hagyja a közösségi tudásbázisok fő erejét, hogy rendkívül dinamikusan fejlődnek, az on-the-fly megoldás kihasználja azt, és mindig a friss adatokkal dolgozik.

További fontos megállapítások, hogy egyik rendszer sem elég flexibilis: futásidőben új komponens beépítése egyáltalán nem lehetséges, nem lehet a feldolgozólánchoz új elemet (kutatást végző modult) illeszteni, szerkezetük szinte mindegyiknek rendkívül statikus és csak arra a célra használhatóak konkrétan, amilyen speciális feladat ellátására kitalálták. Ebből adódóan rendkívül specifikusak és bonyolultak, így újabb kutatások indítása eredményeiket felhasználva nehézkes.

A szoftverek bármely módosítása újrafordítást, rendszerleállást eredményez, ami egy aktívan használt program számára nagy kiesést jelenthet. On-the-fly feldolgozásnál a rendszer leállása kihagyott, nem feldolgozott információkkal jár, így a tudásbázis töredezetté válhat. Ezen okok miatt szükséges egy olyan megoldás, mellyel a rendszer sokkal flexibilisebbé tehető és a fenti problémák áthidalhatóakká válnak.

2. fejezet

Az OSGi keretrendszerről

Az OSGi Alliance [**?**] által fejlesztett OSGi (*Open Services Gateway initiative*) egy Java nyelvű, dinamikus, szolgáltatás orientált komponens modell és keretrendszer, mellyel kiterjeszthető az alap Java nyelven készült programok funkcionalitása. Használatával elérhető válik egy komponens alapú fejlesztés, ahol alkalmazásokat (illetve komponenseket) távolról elérve telepíthetjük, elindíthatjuk, leállíthatjuk, frissíthetjük, vagy törölhetjük anélkül, hogy a teljes alkalmazást leállítanánk és újra elindítanánk. A standard Java-val készült alkalmazásoknál ez egy fontos hiányosság, hiszen az információs rendszerek sok területén nem engedhetőek meg akár a pillanatnyi leállások sem. A flexibilitás és újrafelhasználhatóság követelményeknek tehát nagyon jól megfelel az OSGi keretrendszer, ezért is esett rá a választás.

2.1. Bundle

Az OSGi technológiát [**?**] használó alkalmazások kisebb komponensekre (az OSGI terminológiát használva csomagokra, azaz *bundle*-ökre) vannak bontva. Ezen csomagok elkészíthetőek, lefordíthatóak, telepíthetőek egymástól függetlenül, életciklusukat maga az OSGi keretrendszer felügyeli. A bundle-ök gyakorlatilag nem mások, mint a jól ismert JAR fájlok (Java osztályok, és egyéb erőforrások), azzal a különbséggel, hogy a leíró manifest állományban a szabvány által kiterjesztett módon további fejlécek találhatóak meg. Példát látunk manifest állományra a 2.1. kódrészletben.

A manifest állomány elemei:

Bundle-Name az elkészített bundle emberek számára olvasható neve

Bundle-SymbolicName Java package név, ez azonosítja a bundle-t (az egyetlen kötelező elem)

Bundle-Description a bundle hosszabb szöveges leírása

Bundle-ManifestVersion a bundle által használt OSGi változat verziószáma

Bundle-Version a bundle verziószáma

Bundle-Activator BundleActivator interfészt implementáló bundle-ben lévő osztály, mely a bundle telepítése után elindul

Export-Package más bundle-ök számára elérhetővé tett saját csomagok és verziószámaik listája

Import-Package a bundle fordításához és futtatásához szükséges külső csomagok listája

Private-Package olyan saját csomagok, amelyeket nem teszünk elérhetővé más bundle-ök számára

Listing 2.1. MANIFEST.MF

```
Bundle-Name: Hello World
Bundle-SymbolicName: org.available.helloworld
Bundle-Description: A Hello World bundle
Bundle-ManifestVersion: 2
Bundle-Version: 1.0.0
Bundle-Activator: org.available.helloworld.Activator
Export-Package: org.available.helloworld;version="1.0.0"
Import-Package: org.osgi.framework;version="1.3.0"
Private-Package: org.notavailable.helloworld
```

A bundle-ök egy OSGi példányon belül, azonos JVM-ben futnak. Ennek vannak előnyei (teljesítmény növekedés, kisebb erőforráshasználat, interprocess kommunikációt nem szükséges használni), de hátrányai is (hozzáférési problémák), melyeket az OSGi úgy old meg, hogy minden bundle-höz saját classloader-t rendel.

2.2. Service, Service Registry

A bundle-ök kiajánlhatnak szolgáltatásokat (*service*), melyekre más bundle-ök feliratkozhatnak. Az OSGi specifikáció szerint a szolgáltatások normál Java objektumok, melyek egy adott interfészt implementálva lettek beregisztrálva az OSGi *Service Registry* moduljába.

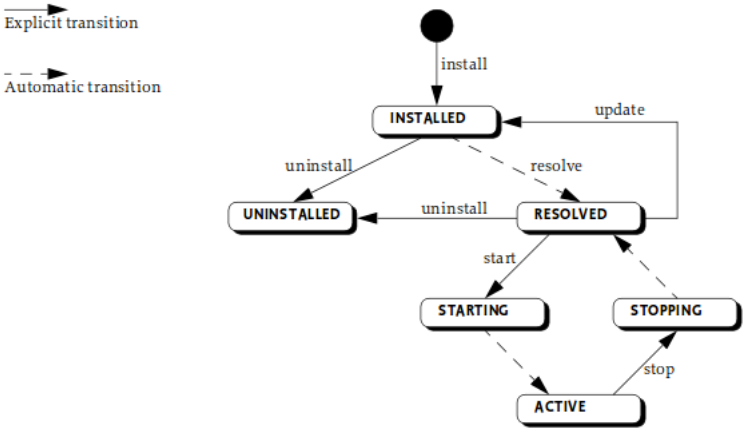
A Service Registry segítségével tudnak tehát a bundle-ök kiajánlani szolgáltatásokat, rajtuk keresztül tudják lekérdezni az elérhető szolgáltatásokat. Ha egy bundle-nek szüksége van egy másik bundle által kiajánlott szolgáltatásra, akkor az OSGi keretrendszer megteremti közöttük a kapcsolatot és a kért szolgáltatást használhatja az adott bundle.

2.3. Életciklus

Állapot neve	Leírás
INSTALLED	A bundle sikeres telepítve lett.
RESOLVED	A bundle-nek minden függősége ki lett elégítve. Kész az elindításra, vagy már le lett állítva.
STARTING	A bundle el lett indítva, de még nincs aktiválva, .start() metódus még nem tért vissza.
ACTIVE	A bundle aktiválva lett és aktív.
STOPPING	A bundle le lett állítva, .stop() metódus még nem tért vissza.
UNINSTALLED	A bundle el lett távolítva, ez az életciklus végállapota.

2.1. táblázat. Bundle életciklus állapotai

Az OSGi keretrendszer dinamikusságát a *Life-cycle* (életciklus) modul szolgáltatja, mely által a bundle-öket futásidőben lehet telepíteni, elindítani, leállítani, frissíteni, eltávolítani más ha-



2.1. ábra. Bundle életciklus (forrás: OSGi Service Platform Release 2 [**?**])

gyományos alkalmazásokkal ellentétben. A bundle-ök futtatása előtt megvizsgálja a keretrendszer a csomag futás idejű függőségeit, és ha olyan kielégítetlen függőségeket talál, melyek szükségesek a bundle futtatásához, akkor nem indítja el a komponenst.

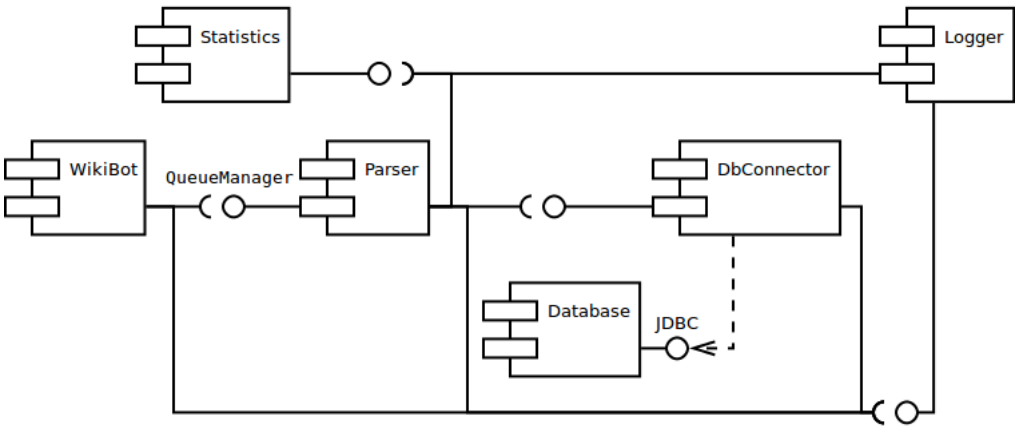
Egy bundle életciklusának állapotai megfigyelhetőek a 2.1. ábrán, valamint az állapotok leírása a 2.1. táblázatban látható.

3. fejezet

Tervezés

Ahogy a 2. fejezetben is láttuk az OSGi keretrendszer megfelel a kért szoftver által támasztott követelményeknek, felhasználásával a rendszer kellően flexibilis lesz. A 1. fejezetben levont tanulságok alapján érdemes megtervezni a rendszert, illetve az ott említett Wikipedia Miner architektúráját fel lehet használni a tervezés során, az alapvető komponensek meghatározásában segíthet.

Első lépésként tehát meghatároztam a leendő rendszer komponenseit (3.1. ábra), mely képes folyamatosan működve a Wikipedia tartalmát rendszerezett formában eltárolni és azt később a rá épülő alkalmazások számára elérhetővé tenni. Mivel a rendszer első indításakor az adatbázisban még semmilyen adatok nem találhatóak, és a Wikipedia frissülő cikkeinek követésével, csak az aktuális változások kerülnek be az adatbázisba, a rendszer indításakor már a Wikipedián korábban közzétett cikkek nem kerülnek be a felépített tudásbázisba. Ezt a problémát úgy lehet a legegyszerűbben megoldani, hogy az on-the-fly feldolgozás mellett a statikus dumpból való adatok importálását is lehetővé tesszük. Ezzel kombináljuk a 1. fejezetben megismert alkalmazások eszköztárát a követelményekben meghatározottakkal, így egy sokkal erősebb eszköz állítható elő.



3.1. ábra. Az alkalmazás tervezett komponensei

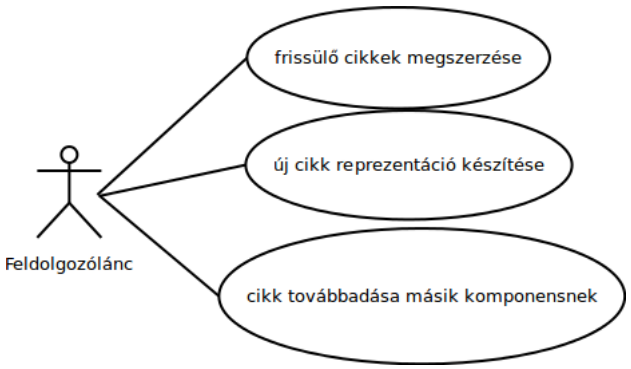
Az eddig szükségszerűen meghatározott komponenseken túl az alkalmazás minőségének javítása érdekében érdemes a komponensek felügyelhetőségét is biztosítani. Ezáltal az alkalmazás működéséből

részletesebb adatok is megfigyelhetővé válnak azon kívül, hogy működik vagy sem a rendszer. A felügyelhetőséget naplózás és különféle metrikák segítségével terveztem lehetővé tenni.

Miután a különálló komponenseket meghatároztam, megkezdődhetett sorban az egyes komponensek megtervezése. Minden komponenst a komponensalapú fejlesztésnek megfelelően egyesével, mint különálló programokat lehetett megtervezni és implementálni. A következőkben bemutatom az egyes komponensek szerepét, alapvető működését és a tervezésük lépéseit.

3.1. WikiBot bundle

Az adatok on-the-fly feldolgozásáért felelős ez a komponens. Ezt a tulajdonságot a frissülő cikkek megszerzésének módjával fogom biztosítani. Az adatok könnyű kezelése miatt minden cikknek egy reprezentációját is ki kell alakítani a programban, amelyet tovább kell adni a következő komponensek számára, további feldolgozásra. Ezek alapján meghatározott use-case-ek láthatóak a 3.2. ábrán.

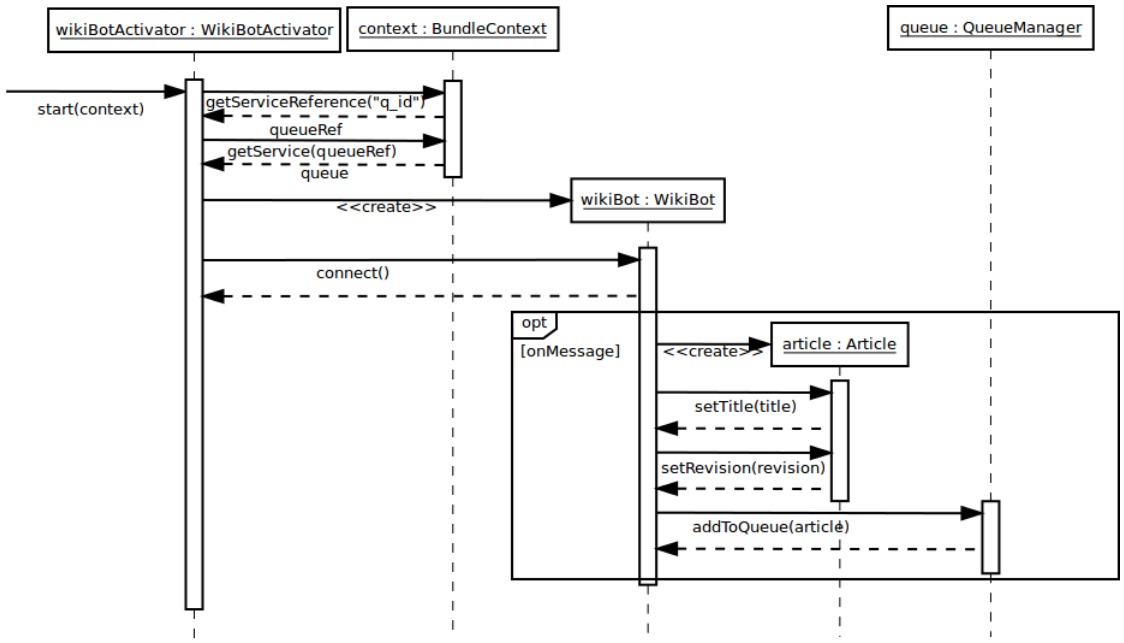


3.2. ábra. A WikiBot komponens használati esetei

Megtervezését egy rövidebb előkutatás előzte meg, mely során keresni kellett valamilyen lehetőséget, amelynek segítségével folyamatosan értesülhet az alkalmazás, ha egy új Wikipedia cikk jelenik meg, vagy frissítenek egyet a hivatalos Wikipedia oldalon. A legegyszerűbb megoldásnak végül egy hivatalos, Wikipedia által üzemeltett IRC csatorna tűnt, ahol folyamatosan publikálják az oldalon frissülő cikkeket.

A tervezett komponens tehát egy IRC Bot kliens lesz, amely a fent említett IRC csatornára lesz feliratkozva. A tervezésnél figyelni kellett arra, hogy miután megszerzi az új cikk szükséges adatait, azokat azonnal tovább kell adnia a következő komponensnek. Más teendőket nem végezhet, működését nem szabad hosszú távon feltartani, mert a gyorsan frissülő cikkek miatt elveszhetnek információk, míg a komponens mással foglalkozik.

A kívánt működést megvalósító szekvencia diagram látható a 3.3. ábrán. A komponens működése egy OSGi BundleActivator implementáció elindulásával kezdődik (`WikiBotActivator` osztály), melyet az OSGi keretrendszer példányosít számunkra, majd elindítja azt. A lehető leggyorsabb működés úgy érhető el, hogy ne tartsuk fel a komponens működését, ha az új cikk megszerzett adatait eltároljuk egy átmeneti tárolóban (`QueueManager` osztály). Ez a queue, ahogy a 3.1. ábrán is látható a Parser komponens egy kiajánlott OSGi szolgáltatása. A `QueueManager` osztály bemutatása a 4.2 alfejezetben folytatódik. A kiajánlott queue referenciáját az OSGi `BundleContext-`



3.3. ábra. A WikiBot tervezett működése

en keresztül lehet megszerezni.

Ezután a wikiBotActivator példányosít egy IRC Bot-ot (`WikiBot`), mely csatlakozik a `.connect()` metódussal a Wikipedia IRC csatornájára. Jelenlegi működés szerint a rendszer az angol nyelvű Wikipedia cikkjeiből épít adatbázist, így belép abba az IRC szobába, ahol az angol nyelvű cikkeket teszik közzé. Az implementált IRC Bot-ban egy eseménykezelőt kell implementálni (`.onMessage()`), ami akkor hívódik meg, ha egy új üzenet érkezik az IRC szobába. Egy példa üzenet figyelhető meg a 3.1. kódrészletben.

Listing 3.1. Példa üzenet az angol nyelvű Wikipedia IRC csatornájából

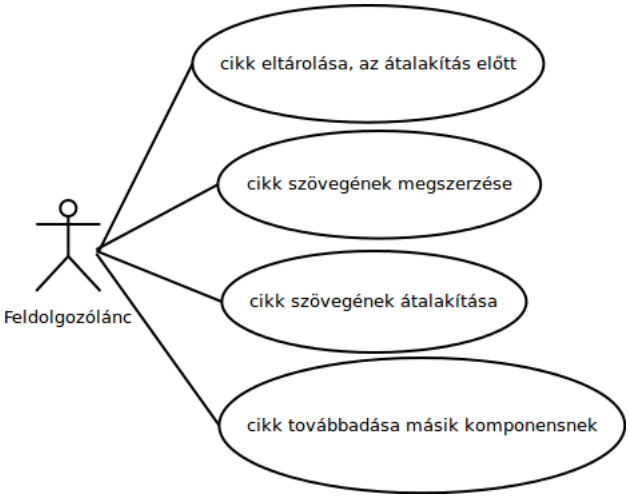
```
(11.16.43) rc-pmtpa: [[History of Vietnam]] http://en.wikipedia
    .org/w/index.php?diff=580135314&oldid=580104406 *
    124.170.231.126 * (+95)
```

Minden egyes beérkezett üzenet alapján, egy új cikk reprezentációt (`Article` osztály) hoz létre, mely később végig fog haladni a teljes feldolgozóláncon. A szükséges adatok a cikk címe, mely dupla szögletes zárójelek között található, valamint az új cikk verziószáma, mely az üzenetben található link `diff` GET paraméterében található. Végül az összeállított cikket a korábban megszerzett `QueueManager`-ben helyezi el.

Az alapvető követelményeken felül a megfigyelhetőséget is érdemes biztosítani, amelyet a `WikiBot` komponensnél naplózással oldottam meg. A `WikiBotActivator` indulásakor a `Logger` komponens (4.4. alfejezet) referenciáját is megszerzi, így a naplózás során a feldolgozóláncban közös, megosztott szolgáltatást használhatnak a komponensek.

3.2. Parser bundle

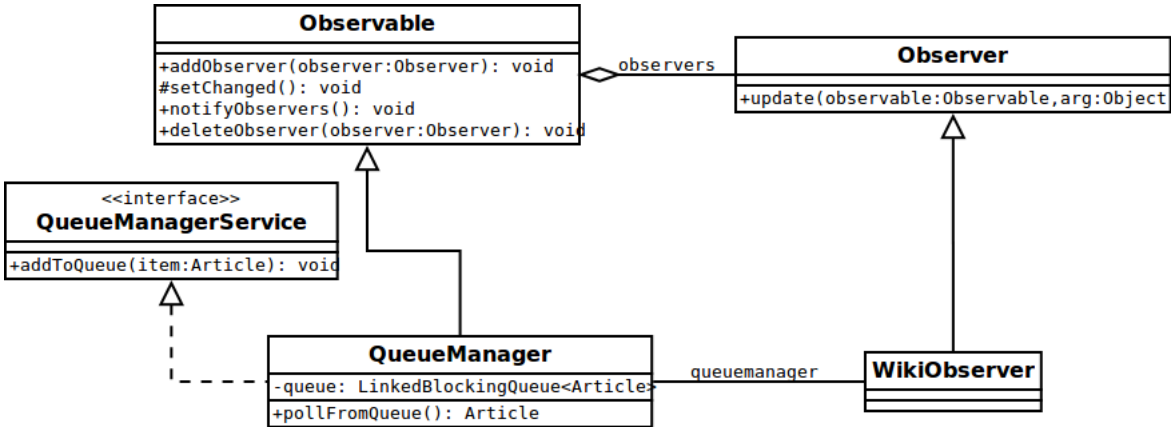
Ebben a komponensben kell lennie valamilyen tároló elemnek, egyfajta queue megoldásnak, melynek szükségessét az előző **??**. alfejezetben fejtettem ki. Ennek a queue megoldásnak szálbiztosnak kell lennie, hiszen egyszerre fog a WikiBot és a Parser komponens dolgozni vele. A módosult, vagy újonnan létrehozott cikk szövegét meg kell szereznie a komponensnek, majd azt át kell alakítania egy meghatározott formátumra. Végül a cikket tovább kell adnia egy másik komponensnek eltárolás céljából. Ezek alapján a use-case diagram elkészíthető (3.4. ábra).



3.4. ábra. A Parser komponens használati esetei

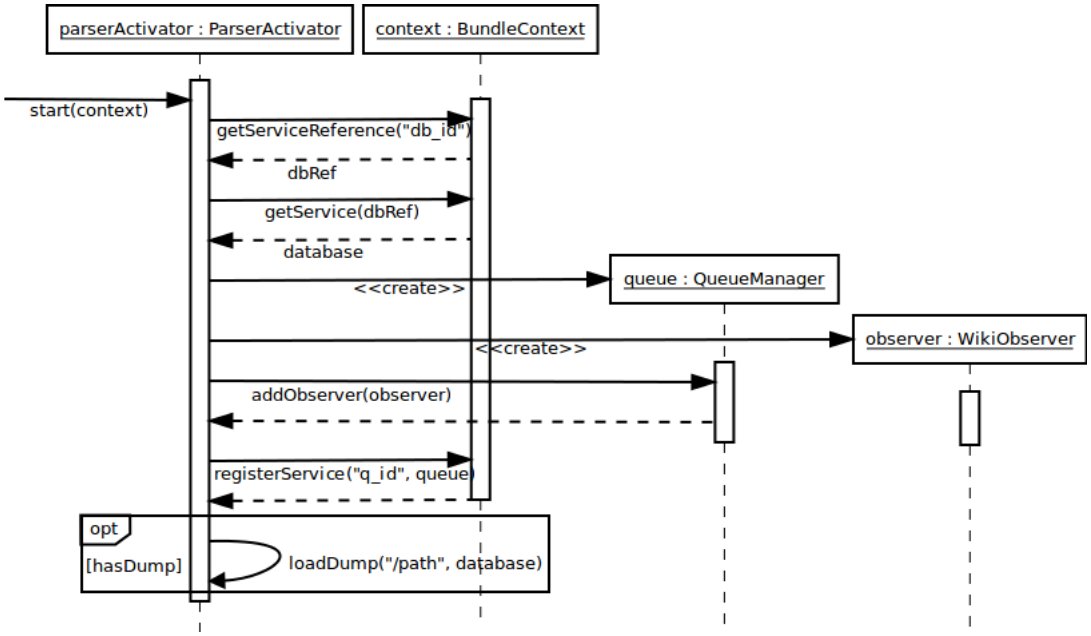
Ahhoz, hogy egy másik komponens el tudjon tárolni a Parser-ben valamit, ahhoz létre kell hozni a Parser komponensben egy szolgáltatást (OSGi service), amelyen keresztül a kommunikáció megtörténhet. Ez az szolgáltatás egyben az egész program mozgatórugója is, hiszen a további use-case-ekben található funkciókat akkor tudjuk végrehajtani, ha legalább egy cikk már el lett tárolva.

Ennél a résznél használtam az *Observer* tervezési mintát: a QueueManager osztály `Observable` lett, míg egy `Observer`-ből származó `WikiObserver` kezelte a tárolóba került új elemeket (3.5. ábra).



3.5. ábra. Observer minta használata a Parser komponensben

A komponens indulásakor először megszerzi a kiajánlott adatbázis szolgáltatás referenciáját, a feldolgozás után itt fogja eltárolni a feldolgozott cikket, minden adatával. Létrehoz ezenkívül egy QueueManager-t, beállít (.addObserver() metódus) számára egy újonnan létrehozott megfigyelőt (WikiObserver osztály), és beregisztrálja a queue-t, mint OSGi szolgáltatást. Végül ha elérhető Wikipedia adatbázis mentés (dump), akkor megkezdi annak feldolgozását is. Az indulás folyamata figyelhető meg a 3.6. ábrán.



3.6. ábra. A Parser komponens indulása

Az Observer tervezési mintának köszönhetően, ha új elem kerül a queue-ba, akkor arról a WikiObserver értesülni fog. Mivel a cikkek feldolgozása jelentős időt vehet igénybe, azokat mindenképpen külön szálakon kell megtenni. Ezt a problémát a *Threadpool* tervezési mintával oldottam meg. A WikiObserver-nek van egy ThreadPool attribútuma, és ha értesül a queue frissüléséről, egy új feladatot fog a ThreadPool-hoz hozzáadni. A ThreadPool mintának megfelelően minden feladat (`WikiWorker`) külön szálon fut.

Ebben a fázisban első lépésben először letöltődnek a hivatalos Wikipedia API-n keresztül a cikkek tartalmai. A Wikipedia a felhasználók számára könnyebben szerkeszthető *Wikitext* nyelven teszi elérhetővé a cikkek tartalmait. A Wikitext egy egyszerű, könnyen használható jelölőnyelv, mely egyértelműen leképezhető a HTML nyelvre, így könnyen készíthetőek vele webes tartalmak.

Azonban ez a Wikitext formátum kevésbé jól feldolgozható, mint például a HTML formátum, így a cikkek szövegét HTML formába kell alakítani. A második fáziban tehát ez az átalakítás történik meg, ha nem volt még újabb verziójú változat az adott cikkből az adatbázisban. Ehhez ugynevezett parser-eket lehet használni, melyből három felcserélhető és különböző előnyökkel és hátrányokkal rendelkező változat is került a feldolgozóláncba.

- Sztakipedia parser: Könnyen testreszabható és általános célú program, mely a MediaWiki Wikitext formátumról tud HTML formátumra átalakítani. Ez a program egy könnyen kiterjeszthető *Visitor* tervezési mintára alapuló szoftver, mely önálló csomagként is használható

a jól felépített API-nak köszönhetően. Használata nagyon egyszerű, használható kimenetet állít elő és sebesség szempontjából is jól teljesít, viszont nagyon nagy méretű fájloknál, például egy teljes Wikipedia dump feldolgozásánál rendkívül sok memóriát használ. Ennek a tulajdonságnak köszönhetően, csak on-the-fly feldolgozás során használható eredményesen ez a parser.

- DumbRegexWikiParser parser: A végletekig leegyszerűsített MediaWiki Wikitext átalakító, mely egy SAX parser implementáció. Az állapotgépes Wikitext feldolgozásnak köszönhetően nem fogyaszt annyi memóriát, mint a Sztakipedia parser, sebesség szempontjából még gyorsabb is az előzőnél, viszont kevésbé használható kimenetet produkál.
- Parsoid parser: Ez az átalakító a MediaWiki által fejlesztett és használt szoftver, mely a Wikitext egy ekvivalens HTML / RDFa kimenetét készíti el, mely automatikus feldolgozásra rendkívül jól használható. Az RDFa szabványban meghatározott attribútumokkal kiegészített HTML kód, sokkal nagyobb jelentéstartalommal bír, mint az alap HTML dokumentum, így a szöveg sokkal használhatóbb lesz további felhasználhatóság szempontjából. A Parsoid egy NodeJS-ben írt szoftver, így felhasználása sokkal nehezebb, mint a korábbi parser megoldásoké, viszont kimenete a célnak leginkább megfelelő, így ez lett az ajánlott átalakító mechanizmus a feldolgozóláncban.

Végül a cikk feldolgozásának végső, harmadik fázisában eltárolódik a cikk a rendszer adatbázisában. Ebbe a fázisba akkor ér el a feldolgozás, ha nem volt újabb verziójú változat az adott cikkből még az adatbázisban. A verziók meghatározása a cikkek verziószáma alapján történik. Ekkor, ha régebbi verziószámú cikk már van az adatbázisban, akkor az a cikket csak frissíteni kell az adatbázisban. Ellenkező esetben az adott cikknek még semmilyen változata nem létezik még az adatbázisban, így azt újonnan kell beszúrni.

3.3. Databaseconnector bundle

Ebben a komponensben tárolódik el a Parser komponensben kinyert cikkhez tartozó minden információ.

3.4. Database

3.5. Logger bundle

3.6. Statistics bundle

4. fejezet

Implementáció, tesztelés és mérések

A komponensek fejlesztésekor Eclipse fejlesztő környezetet használtam, verziókezeléshez pedig a Git szoftvert. Az alapértelmezett Eclipse IDE azonban nem tökéletes eszköz az OSGi alapú fejlesztéshez, ezért a Bndtools [**?**] OSGi fejlesztő keretrendszert Eclipse kiegészítő modulként feltelepítve kezdtem neki a komponensek fejlesztésének.

Teszteléshez a könnyen kezelhető Apache Felix [**?**] OSGi implementációját használtam, mert annak WebConsole eszközével könnyedén kezelhetőek a komponensek menedzselése.

4.1. WikiBot bundle

A WikiBot komponens architektúráját teljes mértékben a fejlesztéskor használt IRC Bot készítő keretrendszerek határozták meg. A kezdeti változatban a PircBot keretrendszert használtam, azonban hosszan tartó használata alatt kiderült, hogy a rendszernek több hibája is van. Hosszabb ideig tartó futáskor ismeretlen eredetű hibák kerültek elő a PircBot könyvtárban, melyek okait a rossz tervezés miatt felderíteni sem volt lehetséges. Ezért egy idő után az újabb és jobban karbantartott verziójára tértem át a PircBot-nak, mely a PircBotX nevet viseli. Ez a váltás azonban csak kis mértékben befolyásolta a fejlesztés menetét, hiszen mindkét keretrendszernél egy esemény alapú architektúra kialakítása szükséges, melyben különféle események esetén implementálni kell a rendszer viselkedését.

A bundle-ök implementációja során a komponensek függőségeinek, melyek külső könyvtárakban voltak, szintén el kellett készíteni a bundle változatait, hogy azok hiánya ne jelentkezzen hibaként az OSGi függőségkezelőjében. Ez a lépés mind a PircBot, mind a PircBotX esetén a külső könyvtárak OSGi komponens formára való alakítását jelentette, melyet a Bndtools nevű eszközzel hajtottam végre.

4.2. Parser bundle

4.3. Databaseconnector bundle

4.4. Logger bundle

4.5. Statistics bundle

5. fejezet

Értékelés, továbbfejlesztési lehetőségek

A félév során rengeteg folyamatosan újabb és újabb technológiákat ismertem meg, ezekben sikerült kellőképpen elmélyedni, így azt hiszem rengeteget tanultam az elvégzett munka alatt. Ezenkívül sikerült egy olyan szoftverrendszer teljes életciklusát végigkövetni és megvalósítani, mely megfelel a kitűzött céloknak és egy használható keretrendszert nyújt azok számára, akik egy ilyen hatalmas tudásbázist akarnak egyszerűen felhasználni, mint a Wikipedia.

Továbbfejlesztési célok között szerepel az adatbázis komponens finomítása, az alkalmazás felügyeleti rendszerének megtervezése, feldolgozási láncok dinamikus előállítása több komponens esetén.

Ábrák jegyzéke

Táblázatok jegyzéke

Forráskódok jegyzéke

Irodalomjegyzék

[1] OSGi Alliance. Osgi service platform release 2. *OSGi Alliance Specifications*, pages 1 – 288, 2001.

[2] Marco Tulio Valente Andre L. C. Tavares. A gentle introduction to osgi. *ACM SIGSOFT Software Engineering Notes*, 33:1 – 5, 2008.

[3] Peter Murray-Rust CJ Rupp-Advaith Siddharthan Simone Teufel Ben Waldron Ann Copestake, Peter Corbett. An architecture for language processing for scientific texts. *In Proceedings of the 4th UK E-Science All Hands Meeting*, 1:1 – 8, 2006.

[4] Neil Bartlett. Bndtools Tutorial. `http://bndtools.org/tutorial.html`, 2011. [Online; hozzáférés 2013.04.25].

[5] Simone Paolo Ponzetto Eduard Hovy, Roberto Navigli. Artificial intelligence, wikipedia and semi-structured resources. *Artificial Intelligence*, 194:1 – 252, 2013.

[6] Ralph Johnson John Vlissides Erich Gamma, Richard Helm. *Desgin Patterns: Elements of Reusable Object-Oriented Software*. Kiskapu Kft., 2004.

[7] Richard S. Hall. Apache Felix OSGi Tutorial. `http://felix.apache.org/site/apache-felix-osgi-tutorial.html`, 2011. [Online; hozzáférés 2013.04.25].

[8] Richard S. Hall. Sztakipédia Parser. `https://code.google.com/p/sztakipedia-parser/`, 2011. [Online; hozzáférés 2013.04.25].

[9] Richard S. Hall2. Sztakipédia Parser. `http://www.jibble.org/pircbot.php`, 2011. [Online; hozzáférés 2013.04.25].

[10] Richard S. Hall3. Sztakipédia Parser. `https://code.google.com/p/pircbotx/`, 2011. [Online; hozzáférés 2013.04.25].

[11] Klaus Berberich Edwin Lewis-Kelham Gerard de Melo Gerhard Weikum Johannes Hoffart, Fabian M. Suchanek. Yago2: Exploring and querying world knowledge in time, space, context, and many languages. *Proceeding WWW '11 Proceedings of the 20th international conference companion on World wide web*, pages 229 – 232, 2011.

[12] Simone Paolo Ponzetto Roberto Navigli. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. *Artificial Intelligence*, 193:217 – 250, 2012.

[13] Iryna Gurevych Torsten Zesch, Christof Müller. Extracting lexical semantic knowledge from wikipedia and wiktionary. *Proceedings of the 6th International Conference on Language Resources and Evaluation*, pages 1 – 7, 2008.

[14] Benjamin Borschinger Cacilia Zirn Anas Elghafari Vivi Nastase, Michael Strube. Wikinet: A very large scale multi-lingual concept network. *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, pages 1 – 8, 2010.

Függelék