BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

An OSGI-based testing and simulation framework for the study of reinforcement learning algorithms

Abstract

The goal of the dissertation is to create a dynamic testing and simulation environment for the evaluation of reinforcement learning algorithms on a remote server using large number of parallel running tests.

The simulation environment is based on the OSGI specification, which defines a dynamic, modularized component model for building complex applications. Previous attempts at creating such a testing environment for an example the `RL-GLUE` project had only partial success. Although it is capable to run and evaluate various tests written in different programming languages, it is already based in obsolete technology and the project was closed a few years ago.

The system I made is capable of evaluating RL algorithms written in JAVA through running various experiments. All this happens with the help of a simulation environment which runs on a remote access server. The client is initiating the test which are written based on a predefined standard, connects to the server which runs the test gets a proper feedback with the help of machine learning algorithms.

The server makes it possible to use a number of predefined simulation environments to perform tests that can be run on the server independently of each other.

The framework enables the monitoring of the experiments, based on a remote method invocation API and through a web interface.

This work is the result of my own activity. I have neither gave nor received unauthorized assistance on this work.

JULY 2015 GÁLL NORBERT

ADVISOR:
JAKAB HUNOR, PH.D, ASSISTANT PROFESSOR

BABEŞ-BOLYAI UNIVERSITY CLUJ–NAPOCA
FACULTY OF MATHEMATICS AND INFORMATICS
SPECIALIZATION: COMPUTER SCIENCE

License Thesis

An OSGI-based testing and simulation framework for the study of reinforcement learning algorithms

SCIENTIFIC SUPERVISOR:
JAKAB HUNOR, PH.D, ASSISTANT PROFESSOR

STUDENT:
GÁLL NORBERT

JULY 2015

UNIVERSITATEA BABEŞ-BOLYAI, CLUJ–NAPOCA
FACULTATEA DE MATEMATICĂ ŞI INFORMATICĂ
SPECIALIZAREA INFORMATICĂ

Lucrare de licenţă

-

CONDUCĂTOR ŞTIINŢIFIC:
LECTOR DR. TANÁR OKOS

ABSOLVENT:
GÁLL NORBERT

IULIE 2015

BABEŞ-BOLYAI TUDOMÁNYEGYETEM KOLOZSVÁR
MATEMATIKA ÉS INFORMATIKA KAR
INFORMATIKA SZAK

Licensz-dolgozat

OSGI technológián alapuló tesztelési és szimulációs keretrendszer megerősítéses tanulási algoritmusok tanulmányozására

TÉMAVEZETŐ:

DR. JAKAB HUNOR, EGYETEMI ADJUNKTUS

SZERZŐ:

GÁLL NORBERT

2015 JÚLIUS

Tartalomjegyzék

1. Bevezető **3**

2. Alapfogalmak **5**

2.1. A megerősítéses tanulás 5

2.2. A megerősítéses tanulás algoritmusok kipróbálásának alap lépései 5

2.3. Alapfogalmak bevezetése 5

3. Felhasznált módszerek és eszközök **9**

3.1. Verziókövetés 9

3.2. Projektmenedzsment 9

3.3. Build rendszer 10

4. Az OSGi keretrendszer **11**

4.1. Az OSGi keretrendszer 11

4.2. Az OSGi architektúra 11

4.2.1. OSGi szolgáltatások 14

5. A rendszer használata **17**

5.1. Teszt indítása 17

6. Következtetés és továbbfejlesztési lehetőségek **18**

1. fejezet

Bevezető

A dolgozat témája a RoboRun projekt bemutatása, mely a megerősítéses tanulási algoritmusok futtatására és tesztelésére biztosít egy egységes szimulációs környezetet egy távoli elérésű szerveren. A dokumentum által megismerhetőek a projekt funkcionalitásai és a hozzá tartozó webes felület.

A RoboRun projekt célja egy olyan dinamikus tesztelési és szimulációs környezet felépítése ahol megerősítéses tanulással kapcsolatos algoritmusok kipróbálására és tesztelésére van lehetőség egy távoli elérésű szerveren. E szimulációs környezet teljesen az OSGi[8] keretrendszerre épül, amely egy dinamikus, modularizált komponens modellt definiál komplex alkalmazások felépítésére. E környezet teljesen egységes és könnyen elérhető.

A tanulás egy nagyon fontos emberi tulajdonság, mely ott van az emberek mindennapjaiban. Hiszen az ember minden nap tanul valami újat, valami új tapasztalattal gazdagodik. A gépi tanulás is az emberi tanuláson alapszik, csak más jelentéssel bír. Mondhatjuk azt, hogy egy olyan folyamat, mely során a tanuló algoritmus paraméterei és belső állapotai változnak, amelyek később meghatároznak egy döntéshozatali stratégiát. Tehát bizonyos tapasztalat alapján, melyet a belső állapotok reprezentálnak, képes a számítógép döntéseket hozni. Ennek a legfőbb nehézsége abban rejlik, hogy véges számú lépés alatt meg kell tanítani a számítógépet arra, hogy egyre jobb döntéseket hozzon végtelen sok lépés közül.

A RoboRun projekt ötlete, nem számít újdonságnak a piacon. Számos hasonló projekt létezik, hasonló funkcionalitásokkal. A RoboRun projekt szempontjából az Rl-Glue [1] projektet érdemes kiemelni, hiszen ez szolgált a RoboRun projekt alapjául és számos funkcionalitását is felhasználtuk a projekt során. Az Rl-Glue[10] projekt szerzői Brian Tanner és Adam White. E projekt eredetileg C++ ban íródott, viszont van teljesen Java- ban megírt változata is. Az Rl-Glue egy nyelv független környezet a megerősítéses tanulási algoritmusok tanulmányozására. Kétféle protokollt kínál, az úgymond külső- illetve belső módokat. A külső mód teljesen socketeken keresztül végzi a kommunikációt, míg a belső mód az teljesen lokálisan. Ez által a belső mód sokkal gyorsabb működést eredményez. A projekt 2010- ben lezárult, viszont teljesen nyílt forráskódú, mindenki számára elérhető és használható napjainkban is. Néhány hasonló projekt: RL Toolbox[2], ClSquare[3], PIQLE[4].

A RoboRun projekt az Rl-Glue projektet tovább gondolva és funkcionalitásait felhasználva, napjaink technológiáin alapszik. Egy dinamikus és egységes környezetet biztosít, mindezt úgy, hogy a rendszer

1. http://glue.rl-community.org/wiki/Main_Page
2. http://www.igi.tu-graz.ac.at/gerhard/ril-toolbox/general/overview.html
3. http://ml.informatik.uni-freiburg.de/research/clsquare
4. http://piqle.sourceforge.net/

teljesen moduláris, folyamatosan és könnyen változtatható, illetve bővíthető. Mindezek mellett nagyon jól megvalósítja a komponensek egymástól való elválasztását. A projekt teljesen Java alapokon nyugszik, felhasználva az OSGi platformot.

A fő változtatásokat főként a dinamikusság, a modularitás illetve a könnyed bővíthetőség foglalja magában. Az Rl-Glue - hoz képest a teljes architektúrát újra kellett gondolni és teljesen újra felépíteni a projektet az új architektúrára, annak érdekében, hogy helyt álljon az OSGi környezetben. Az új architektúra által a projekt, sokkal átláthatóbb, könnyebben kezelhető és teljesen megvalósítja a komponensek egymástól való elválasztását. E mellett a RoboRun projekt egy nagy előnye, hogy nagyon kis erőforrásra van a felhasználónak szüksége még komolyabb tesztek futtatásánál is, hiszen a fő logikát, tehát az erőforrás igényes részeket mind a távoli elérésű szerver futtatja. A projekt lehetőséget nyújt a felhasználók számára, egy webes felületen keresztül arra, hogy megtekinthessék az aktuálisan futó teszteket, illetve, a már lefuttatott teszteredmények is megtekinthetőek.

A dolgozat hat fejezetből áll. Az első fejezet röviden bemutatja a megerősítéses tanulást néhány világbeli példán keresztül, majd bemutatásra kerül néhány alap fogalom illetve ezek felhasználása a projekt során.

A második fejezet a projekt által felhasznált módszereket és eszközöket mutatja be, illetve azt, hogy ezek pontosan milyen szerepet játszódtak a projekt megvalósítása során.

A harmadik fejezet a RoboRun projekt alapjául szolgáló OSGi keretrendszert mutatja be, ismerteti ennek architektúráját, illetve azt, hogy miért esett e keretrendszerre a választás. Megemlít más eszközöket és technológiákat is melyek felhasználásra kerültek. E fejezet kitér arra is, hogy a projekt, hogyan használja a megerősítéses tanulási kísérletek lebonyolítására.

A negyedik fejezet részletezi a projekt által felhasznált technológiákat, majd tárgyalja a rendszer felépítését, ismerteti a szerver oldali architektúrát.

Az ötödik fejezet a rendszer használatát és működését mutatja be néhány egyszerűbb példán keresztül. Részletesen kitér a rendszer által nyújtott funkcionalitásokra így ezen fejezet felhasználói dokumentációnak is tekinthető.

A hatodik fejezet a RoboRun projekt továbbfejlesztési lehetőségeit részletezi.

A RoboRun projekt sikeres elkészítéséért köszönet illeti a projektvezetőt.

2. fejezet

Alapfogalmak

Összefoglaló: *E fejezet célja bemutatni röviden a megerősítéses tanulást és ezen algoritmusok alap lépéseinek bemutatását, illetve a RoboRun projekttel kapcsolatos néhány alap fogalom bevezetése és ezek használata a projekt során.*

2.1. A megerősítéses tanulás

2.2. A megerősítéses tanulás algoritmusok kipróbálásának alap lépései

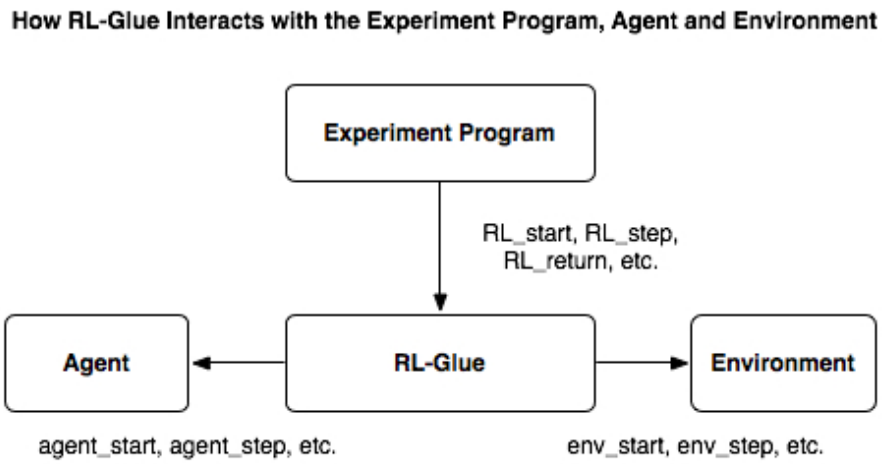
2.3. Alapfogalmak bevezetése

A projekt során a szerző az Rl-Glue projekt elveit követte, felhasználva annak funkcionalitásait. A három alap komponens mindkét projekt esetén az `Agent`[1], `Environment`[2] és az `Experiment`[3]. Ezen komponensek egymással való interakciója révén nyílik lehetőségünk futtatni illetve tesztelni a megerősítéses tanulási algoritmusokat.

Az Agent komponens valójában a tanulási algoritmus, amely kiszabja a feladatokat és az ezekre vonatkozó megszorításokat egy adott iterációra vonatkozóan. Az Agent jutalmat(reward) kap minden egyes iteráció után arra vonatkozóan, hogy a probléma megoldásának szempontjából mennyire volt hatékony a kiszabott feladat, illetve az erre vonatkozó megszorítás. Mivel nem tudhatja az algoritmus, hogy melyik a helyes módszer a probléma megoldására, ezért találgatnia kell. Időnként új cselekvéseket is kell próbálnia, majd az ezekből megszerzett tudást, ami esetünkben a jutalom, optimális módón felhasználnia a következő cselekvés meghatározására.

Az Environment komponens feladata végrehajtani az Agent komponens által meghatározott feladatokat és az ezekre vonatkozó megszorításokat az adott problémára. A végrehajtás során következtetéseket(observation) von le minden egyes állapotról. Majd ezen következtetések alapján jutalmakat(reward) határoz meg. Mivel bizonytalan a környezet, valami becslést kell alkalmaznia a jövőre nézve, így kezdetben, lehet, hogy egy jó lépésért nem kapjuk meg a megfelelő jutalmat. Viszont minél jobban megismerjük a környezetet, annál pontosabb lesz egy lépésért vagy lépés sorozatért járó jutalom. A jutalom egy számban fejezhető ki, amely egy adott intervallumban mozog. Ha az intervallum felső határához közelít a szám akkor pozitív visszajelzést kaptunk az adott lépés vagy lépes sorozat után, amennyiben az intervallum alsó határához közelít a szám, negatív a visszajelzés.

1. Agent - magyarul ügynök
2. Environment - magyarul környzet
3. Experiment - magyarul kísérlet



2.1. ábra. Rl-Glue projekt komponensek közti kommunikációja:
http://rl-glue.googlecode.com/svn/trunk/docs/html/index.html

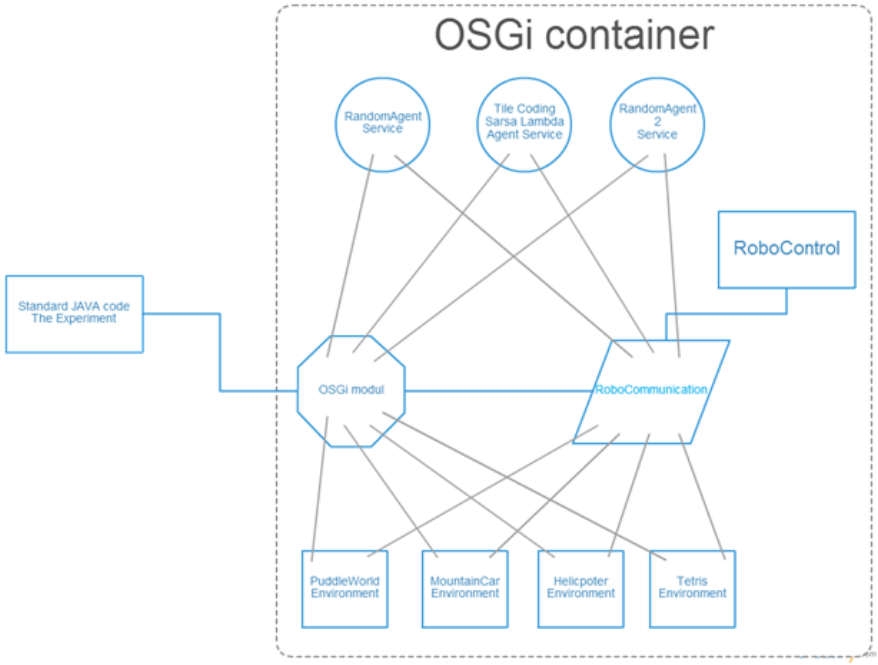
Az Experiment komponens irányítja a teljes kísérlet végrehajtását. E komponens nincs direkt kapcsolatban az Agent és az Environment komponensekkel. Van köztük egy köztes réteg, amely végzi a kommunikációt e három komponens között. Az Experiment komponensben van meghatározva a lépések száma egy adott iterációban, illetve az iterációk száma is. Fontos azon szerepe is az Experiment komponensnek, hogy a végső eredményeket ő kapja meg az Agent illetve az Environment komponensektől a köztes rétegen keresztül. A fent említett köztes réteg az Rl-Glue projekt esetén az úgynevezett RL-Glue mely a teljes kommunikáció lebonyolítását végzi a komponensek között illetve létrehozza a hálózati kommunikációhoz szükséges objektumokat.

A RoboRun projekt esetén az RL-Glue komponens helyét felváltja a RoboControl komponens, viszont a funkcionalitások nagy része megmarad vagy csak részben változik. Például a RoboRun projekt esetén nincs szükség ezen a szinten beállítani a hálózati kommunikációt.

Az Agent, Environment, Experiment és a RoboControl hasonló módon működnek mint az Rl- Glue projekt esetén, viszont a RoboRun projektben az Experiment kivételével, minden komponens egy szolgáltatás az OSGi konténerben, amely egy távoli GlassFish[6] szerveren fut. Az Experiment komponens és az OSGi konténer közötti kapcsolat megteremtéséért egy OSGi modul a felelős. E modul ismeri az összes konténerbe telepített Agent illetve Environment komponenst.

A konténerben egyszerre több előre definiált Agent és Environment lehet telepítve, ezek száma nincs korlátozva. Bármikor módosítható, törölhető vagy teljesen új Agent és Environment is hozzáadható a konténerhez anélkül, hogy a szervert meg kellene állítani vagy újra kellene indítani.

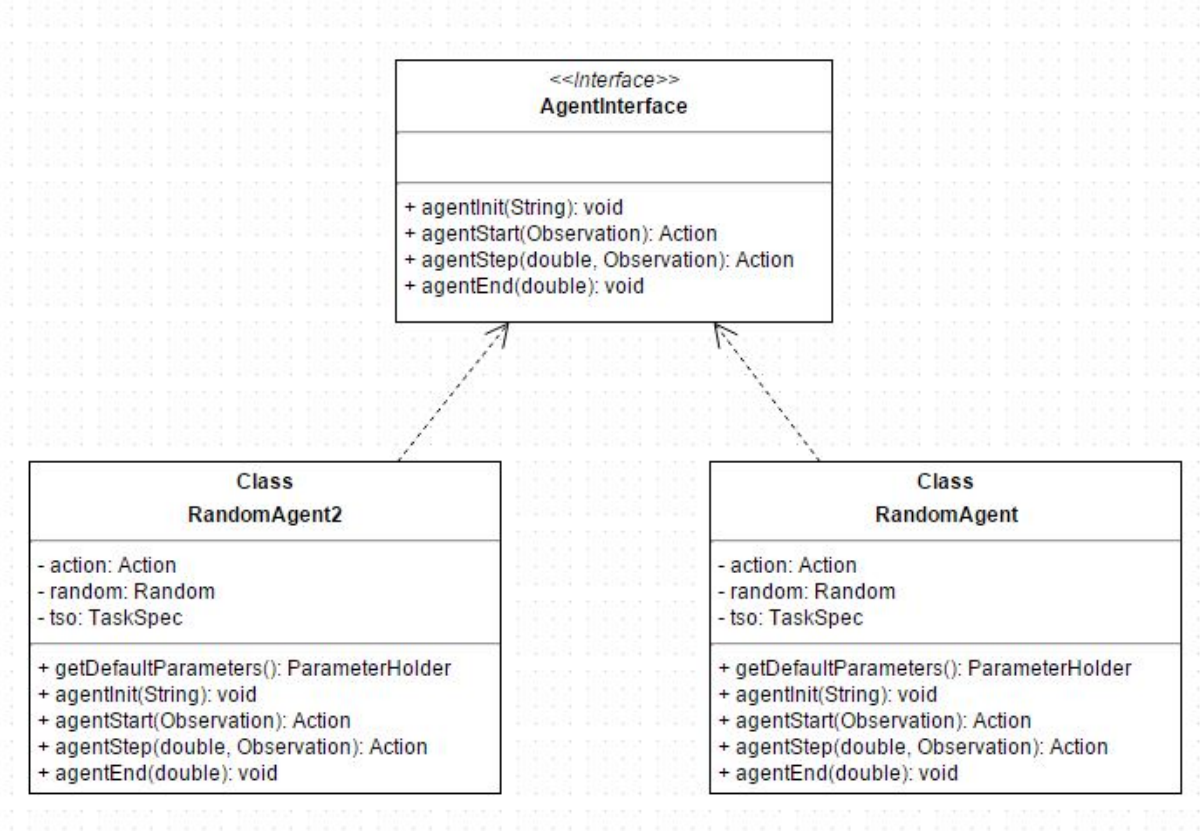
Egy kísérlet futtatása során kliens oldalon az Experiment komponens meghatározza a szükséges lépések számát iterációnként és az iterációk számát, majd kapcsolatba lép az OSGi modullal. Az OSGi modul a fent említett módon, ismer minden olyan Agent - et és Environment - et, amely telepítve van a konténerbe. Ezek közül választhat az Experiment komponens, hogy melyiket szeretné használni. Tehát kiválaszthatja azt, hogy melyik Environment - hez milyen Agent -t szeretne használni a kísérlet során.



2.2. ábra. A RoboRun projekt alap komponensek közti kommunikáció:

Ezek az információk mind az OSGi modulon keresztül jutnak el a szervertől a kliensig. Amint az Experiment komponens meghatározta a kívánt Agent -t és Environment -t, az OSGi modul közvetíti ezt a RoboCommunication komponens fele, mely lekérdezi a kiválasztott Agent és Environment szolgáltatásokat. Ezek folyamatos interakcióba kerülnek egymással melyet a RoboControl komponens irányít. Amint véget ér a kísérlet az Experment komponens az OSGi modul segítségével megkapja a kísérlet eredményét.

Minden `Agent` implementálja az `AgentInterface` -t és minden `Environment` implementálja az `EnvironmentInterface` -t.



2.3. ábra. Az Agent osztály UML diagramja

3. fejezet

Felhasznált módszerek és eszközök

Összefoglaló: *Minden nagyobb projekt esetén a fejlesztőknek szükségük van arra, hogy a megfelelő felkészültségük és találékonyságuk mellett, figyelmet fordítsanak a hatékony munkára is. Ehhez szükségük lehet különböző verziókövető rendszerek használatára, projektmenedzsment eszközökre és build rendszerekre.*

3.1. Verziókövetés

Napjainkban egyre nagyobb szükség van arra, hogy egy projekt esetén a munka könnyedén megosztható és hordozható legyen a fejlesztők közt. E mellett nagyon fontos a fejlesztési folyamat monitorizálása. Ezen technológiák nélkül szinte elképzelhetetlen a szoftverfejlesztés, úgy csoportos környezetben mint egyedül.

E célra fejlesztették ki a verziókövető rendszereket és a projektmenedzsment eszközöket melyek által könnyedén megoszthatóvá válik a fejlesztői munka és folyamatosan ellenőrizhető a fejlesztés folyamata.

A verziókövető rendszer által folyamatosan nyomon követhető a projekt fejlődése és ellenőrizhető az egyénenkénti haladás is. Könnyed visszaállítási lehetőséget biztosít arra az esetre, ha valami történne a lokális gépünkön tárolt forrás állományokkal vagy ha bármi hiba történne a fejlesztés során ami visszaállítást igényel. Legnagyobb haszna a verzió követő rendszereknek, az olyan projekteknél van, amelyet több fejlesztő fejleszt egyszerre. Hiszen általában ilyenkor a projekt teljes forrásállománya egy központi tárolóban van elhelyezve ahová mindenki beteszi a változtatásait. Így nagyon egyeszűen követhető, hogy melyik fejlesztő milyen fázisban tart.

A RoboRun projekt fejlesztése során a forrásállományok tárolására és a fejlesztés nyomkövetésére felhasznált verziókövető rendszer a Git[5], amely nyílt forráskódú és teljesen ingyenes. Webes felületet biztosít a tároló megtekintésére. Könnyedén megoszthatóak a forrásállományok. A projekt szerzője által használt kliensalkalmazás a TortoiseGit[12]. A TortoiseGit szintén ingyenes szoftver, melyet szükséges telepíteni. Használata egyszerű. A konzol mellett, rendelkezik egy felhasználóbarát grafikus felülettel is, mely még inkább megkönnyíti a használatát.

3.2. Projektmenedzsment

"A projektmenedzsment az erőforrások szervezésével és azok irányításával foglalkozó szakterület, melynek célja, hogy az erőforrások által végzett munka eredményeként egy adott idő- és költségkereten belül sikeresen teljesüljenek a projekt céljai." (forrás: http://hu.wikipedia.org/wiki/Projektmenedzsment)

A projektmenedzsment eszközök fő célja tehát, hogy a fejlesztők a specifikáció által meghatározott feladatot adott idő - és költségkereten belül sikeresen tudják teljesíteni. Ennek érdekében számos hasznos funkcionalitást biztosítanak. Ilyen funkcionalitások például, különböző feladatkörök kiosztása, különböző feladatok kiosztása, egy adott folyamatra szánt idő meghatározása, a fejlesztő által eltöltött munkaidő egy adott rész megvalósításával. Mindezek mellett kommunikációs lehetőséget biztosít a fejlesztők között. Lehetőség ad feltölteni dokumentumokat, diagramokat, segédanyagokat a projekthez, ez által megkönnyítve a fejlesztők munkáját.

A RoboRun projekt fejlesztése során alkalmazott projektmenedzsment eszközként a Redmine[9] webes menedzsment eszköz szolgált. A Redmine egy teljesen nyílt forráskódú és platform független projektmenedzsment rendszer. Egyszerű és letisztult felülete révén könnyen kezelhető. Rengeteg funkcionalítást nyújt, mely nagy segítség lehet a különböző projektek fejlesztése során. A Redmine által nyújtott néhány fontosabb funkcionalitás: naptár, e-mail értesítés, szerepkör szerinti hozzáférés, wiki és fórum, pluginok engedélyezés, adatbázisok támogatása, stb.

3.3. Build rendszer

A build rendszereket többnyire projektek menedzselésére és a build folyamat automatizálására alkalmazzák.

A RoboRun projekt fejlesztése során alkalmazott build rendszer a Maven[7], amelyet Jason van Zyl készített 2002-ben. A Maven egy nyílt forráskódú, platform független eszköz. Leggyakoribb felhasználása a Java nyelvben írt projektek esetében történik. A Maven konfigurációs modellje XML alapú, e mellett bevezetésre került a POM(Project Object Model). A POM az adott projekt szerkezeti vázának teljes leírását tartalmazza és a modulokat azonosítókkal látja el. Tehét a POM egy projekt leírását tartalmazza és a projekthez tartozó összes függőség listáját. Ezen függőségeket a Maven a saját központi tárolójából tölti le a projekt buildelése során. A POM esetén a lépéseket céloknak nevezik. A célok lehetnek előre definiáltak, mint például a forráskód csomagolása és fordítása vagy lehetnek a felhasználó által meghatározott célok. Mindezt a pom.xml állomány által valósul meg, amely tartalmazza ezeket az információkat.

A RoboRun projekt esetén a Maven build eszköz legfontosabb szerepe a függőségek, célok és pluginok kielégítése a build folyamat során, hiszen a Maven saját függőség kezelő rendszerrel rendelkezik, amely a build - elés során letölti a központi tárolóból az előre megadott függőségeket és elhelyezi a lokális tárolóban, ahonnan a jövőben használni fogja. E mellett a Maven lehetőséget nyújt a projekt moduljainak azonosítására a groupID, az artifactID és a verzió szám révén. A groupID logikai csoportokba szervezi a komponenseket, az artifactID minden komponenst egyedi azonosítóval lát el és a verzió az éppen aktuális verziószámot takarja a komponensek esetén.

4. fejezet

Az OSGi keretrendszer

Összefoglaló: *E fejezet célja bemutatni az OSGi keretrendszert, illetve annak architektúráját. Bemutatja, hogy a RoboRun projekt miért használja az OSGi keretrendszert. Végül egy általános leírást ad arról, hogy a RoboRun projekt, hogyan használja az OSGi keretrendszert a megerősítéses tanulási kísérletek futtatására és tesztelésére.*

4.1. Az OSGi keretrendszer

Az OSGi -t eredetileg arra fejlesztették ki, hogy home gateway - ként működjön. Ez azt jelenti, hogy a home gateway kapcsolatban áll egy szolgáltatóval és a felhasználók által kifizetett szolgáltatásokhoz biztosít elérést. Tehát a szolgáltató kezében van a teljes menedzselés joga, a felhasználó csak használja az adott szolgáltatásokat.

Az OSGi keretrendszer alapötlete a szolgáltatás orientált architektúrára[11] vezethető vissza. A szolgáltatás orientált architektúra olyan szolgáltatásokat és komponenseket biztosít, amelyek eleget tesznek egy bizonyos szabványnak, biztonságosak és egymáshoz lazán kapcsolódnak. Ezen komponensek folyamatosan változtathatóak és újra felhasználhatóak.

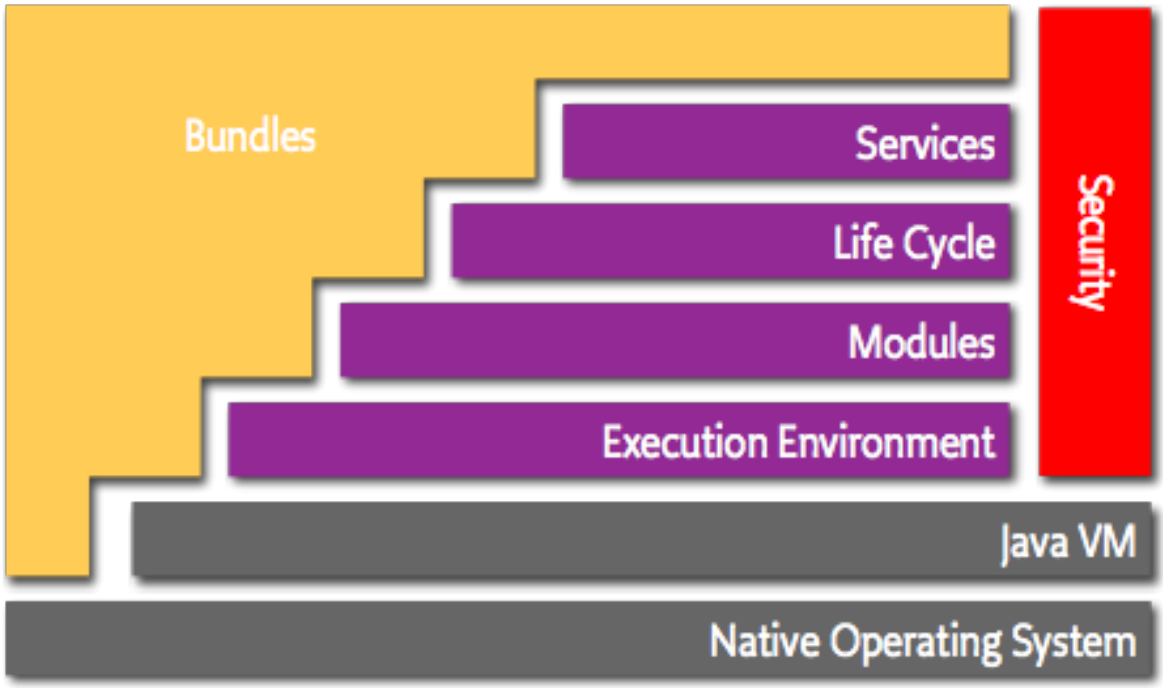
Az OSGi keretrendszer egy olyan keretrendszer, mely a Java nyelv fölött fut. Az OSGi jelentése, Open Service Gateway Initiative. E keretrendszer célja bővíthető Java alkalmazások fejlesztésének a támogatása. Teljesen dinamikus környezetet biztosít, hiszen képes kezelni a csomagok futás idejű megjelenését és eltűnését a nélkül, hogy a felhasználó bármit is észrevenne ebből. Lehetőséget nyújt különböző szolgáltatások definiálására, amelyek folyamatosan bővíthetőek, változtathatóak, szintén futás időben. Mindezek mellett nagyon jól megvalósítja a komponensek egymástól való elkülönítését. Többféle implementációja ismert az OSGi keretrendszernek, például az Apache Felix[1] vagy az Eclipse keretrendszer alapjául szolgáló Eclipse Equinox[4].

4.2. Az OSGi architektúra

Az OSGi keretrendszer különböző eszközöket biztosít a szolgáltatások építése érdekében. Ilyen alap eszközök például a batyuk[2].

Bundles(batyuk vagy kötegek)

A batyukat az OSGi keretrendszer alapjának tekinthetjük. Általánosan három részből tevődnek össze: Java - kód, statikus erőforrások(pl.: képek) illetve leíró állomány vagy MANIFEST.MF - fájl. A programegységek az OSGi keretrendszerben batyuként kerülnek telepítésre. A batyuk rendelkeznek néhány



4.1. ábra. OSGi architektúra http://www.osgi.org/Technology/WhatIsOSGi

fontos tulajdonsággal, ilyen tulajdonságok, hogy minden batyuhoz megadhatóak különböző jogok, a batyuk életciklusainak változásai különböző eseményeket generálnak, melyre feliratkozhatnak más batyuk, a batyuk lehetnek futtathatóak, amennyiben implementálják a `BundleActivator` osztályt, viszont ez nem kötelező. E mellett a batyuk egy nagyon fontos tulajdonsága az, hogy képesek szervizeket regisztrálni, amelyek által más batyuk számára elérhetővé vállnak.

A leíró állomány által értelmezhető a batyu tartalma:

```
Bundle-Name: rmiExperiment
Bundle-SymbolicName: edu.bbte.rmiExperiment
Bundle-Version: 1.0.0
Bundle-Vendor: GALLNORBERT
Bundle-Activator: edu.bbte.rmiExperiment.Activator
Export-Package: edu.bbte.rmiExperiment
Import-Package: edu.bbte.packages,
 org.osgi.framework;version="1.3.0"
```

A **Bundle-Name**- től a **Bundle-Vendor**- ig a batyuról tárolt információk találhatóak, a **Bundle-Activator**- azt az osztályt tartalmazza, amelyik elindul a batyu telepítésekor és annak törlésekor leáll. Az **Export-Package** a batyu által közzétett csomagokat tartalmazza, míg az **Import-Package** azon csomagokat tartalmazza, amelyekre a batyunak szüksége van a futás során. Természetesen a MANIFEST.MF - állomány más elemeket is tartalmazhat, illetve a példában lévők sem kötelezőek mint. Például a **Bundle-Activator** címkét nem kötelező megadni, hiszen nem minden batyunak van szüksége `Activator` osztályra.

Példa Activator osztályra:

```
public class Activator implements BundleActivator  {

    public void start(BundleContext context) throws Exception {
```

```
        // ....
    }
    public void stop(BundleContext context) throws Exception {
        // ....
    }
}
```

A batyu telepítésekor az OSGi keretrendszer példányosítja az `Activator` osztályt és meghívja a start() metódusát automatikusan. A start() metódus megkap egy `BundleContext`- re mutató referenciát mely által új szervizeket lehet regisztrálni és lekérdezni, a keretrendszer különböző eseményeire lehet feliratkozni, batyukat lehet lekérdezni.

A batyuk rendelkeznek a `MANIFEST.MF` állomány révén az export - import mechanizmussal. Ez által a batyuk közzétehetik az osztályaikat más batyuk számára. Alapértelmezetten minden batyuban lévő csomag rejtett a többi batyu elől. Azokat a csomagokat amelyeket közzé szeretnénk tenni más batyuk számára az **Export-Package** címkével tehetjük meg és az **Import-Package** címke segítségével kérhetjük le azon csomagokat amelyekre szükségünk van más batyukból, természetesen csak akkor, ha ezek publikussá vannak téve a batyu által. A batyuk esetében a csomagfüggőség mellett beszélhetünk batyufüggőségről(Require-Bundle) is. Ezt akkor használják, amennyiben szükség a függőséget csak a teljes batyu képes kielégíteni.

Egy batyuban négyféle csomag érhető el:

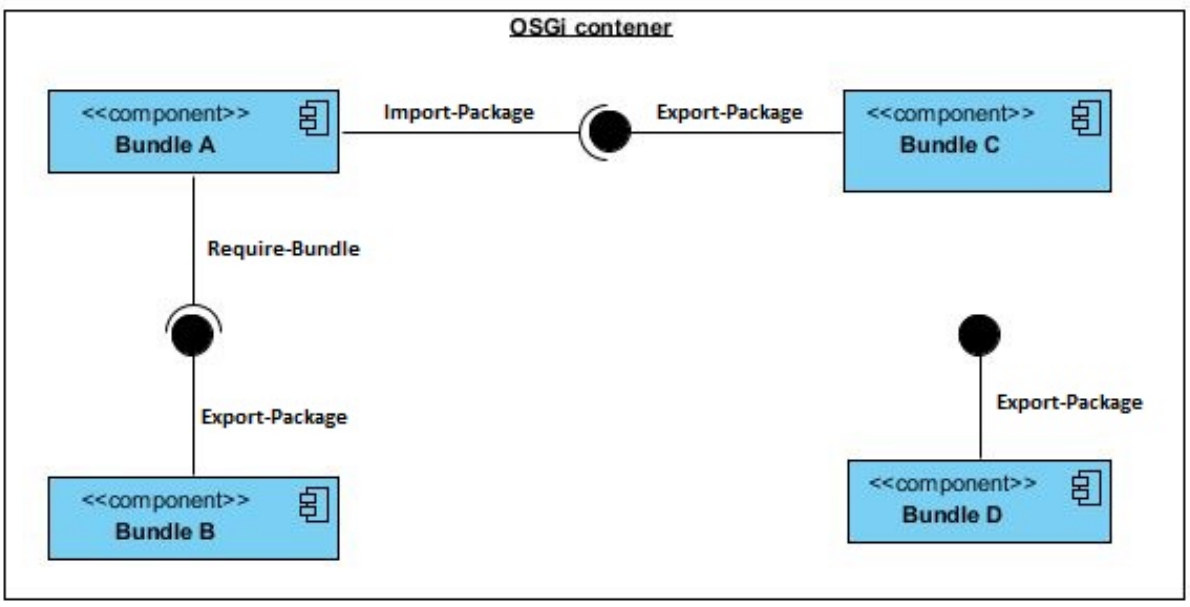
A batyu által létrehozott csomagok

Az **Import-Package** által megadott csomagok

A **Require-Bundle** által megadott batyu összes publikus csomagja

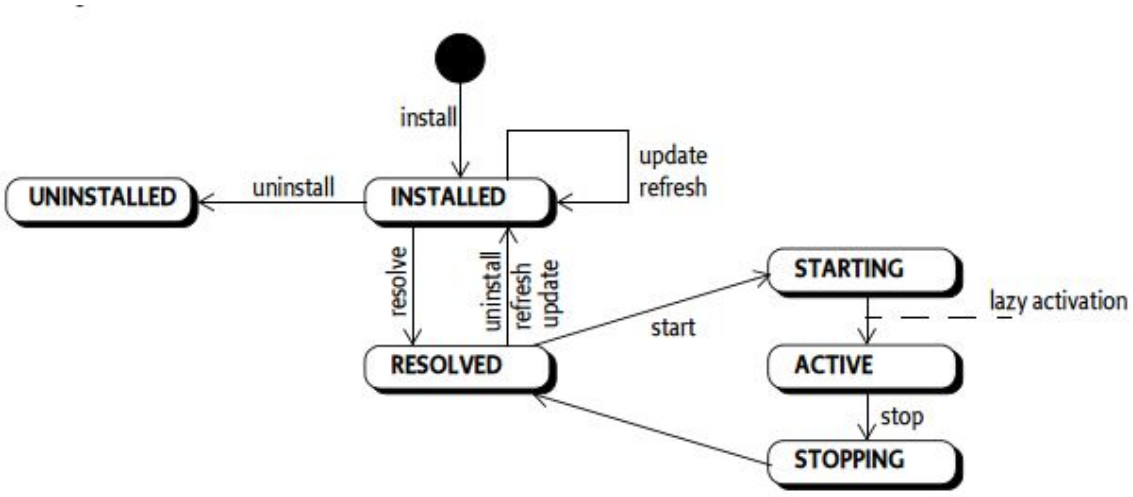
A Java összes függvénykönyvtára

A batyuknak vannak különböző állapotaik, mely által meghatározható, hogy éppen mi történik velük.



4.2. ábra. Batyu által elérhető csomagok

Ezen állapotok végig kísérik a batyut a telepítés pillanatától egészen a törlésükig. Ezen állapotokat a batyu életciklusának is szokták nevezni.



4.3. ábra. Batyu életciklusa https://osgi.org/download/r6/osgi.core-6.0.0.pdf

Istalled vagy **Installált** állapot: A batyu sikeresen installálásra került az OSGi keretrendszerben
Resolved vagy **Feloldott** állapot: A batyu export illetve import függőségei sikeresen ki vannak elégítve és az általa kiajánlott csomagok is használhatóak a többi batyu számára
Starting vagy **Indulás** állapot: A batyu `Activator` osztályának start() metódusa meghívásra került de még nem tért vissza
Active vagy **Aktív** állapot: A batyu teljesen aktív a konténerben és használható.
Stopping vagy **Leállás** állapot: A batyu stop() metódusa meghívásra került de még nem tért vissza
Uninstalled vagy **Törölt** állapot: A batyu törlésre került és csak akkor használható újra ha újra telepítik a rendszerbe

4.2.1. OSGi szolgáltatások

Az OSGi architektúra egy másik nagyon fontos építő eleme a szolgáltatások. A szolgáltatások által kódrészleteket lehet elérhetővé tenni, illetve ez biztosítja a batyuk közti dinamikus kommunikációt. Jól definiálja az együttműködés modellt:"publish-find-bind". A szolgáltatás egy közönséges java objektum, amely támogatja a dinamikus, futásidejű változásokat. Ez azt jelenti, hogy futási időben jelenhetnek meg szolgáltatások, melyeket azonnal használatba lehet venni, illetve ezek törlésre is kerülhetnek, szintén futási időben. A szolgáltatások elérése érdekében az OSGi konténer egy szolgáltatás tárolót biztosít. Minden szolgáltatást ide kell beregisztrálni és majd innen lehet kikérni. Minden szolgáltatás kötelező módon egy interfészt implementál és ezen interfész nevén kell regisztrálva legyen. Fontos kiemelni azt, hogy az interfész és az interfész implementációja nem kell egy batyuban legyenek. Az interfészt tartalmazó batyu

közzéteszi a megfelelő csomagot, majd ezt a csomagot importálja a szolgáltatás implementációt tartalmazó batyu. Ez biztonság szempontjából is igen fontos tulajdonság lehet. A másik fontos előnye ennek az architektúrának az, hogy így több szolgáltatás is implementálhatja ugyanazt az interfészt. Abban az esetben, ha több szolgáltatás implementálja ugyanazt az interfészt akkor használni kell egy egyedi azonosítót. A szolgáltatások regisztrálását a szolgáltatás tárolóba a `ServiceRegistration` komponens által lehet megvalósítani.

```
public class Activator implements BundleActivator  {

        private ServiceRegistration serviceRegistration;

    public void start(BundleContext context) throws Exception {

        serviceRegistration  = context.registerService(Example.class.getName(), new
            ExampleImpl(), null);
    }

    public void stop(BundleContext context) throws Exception {

        environmentServiceReg.unregister();
    }

}
```

A fenti példa esetén az `Activator` osztály implementálja a `BundleActivator` interfészt, mely két metódussal rendelkezik, a `start(BundleContext context)` és a `stop(BundleContext context)` metódusokkal. A `BundleContext`- által új szolgáltatásokat lehet regisztrálni és lekérdezni. A `context.registerService(Example.class.getName(), new ExampleImpl(), null))` metódus az `Example` interfész neve által beregisztrálja az OSGi szolgáltatás tárolójába az `ExampleImpl` szolgáltatást. Az `ExampleImpl` osztály implementálja az `Example` interfészt.

Abban az esetben ha több szolgáltatás implementálja ugyanazt az interfészt, szükség van az egyedi azonosító használatára.

```
public class Activator implements BundleActivator  {

        private ServiceRegistration serviceRegistration;

    public void start(BundleContext context) throws Exception {

        Hashtable<String, String> dictionary = new Hashtable<String, String>();

        dictionary.put(Constants.SERVICE_DESCRIPTION, "This is a Bundle");
        dictionary.put("Name", "ExampleBundle");

        serviceRegistration  = context.registerService(Example.class.getName(), new
            ExampleImpl(), dictionary);
    }

    public void stop(BundleContext context) throws Exception {

        environmentServiceReg.unregister();
    }

}
```

Megadható a `registerService()` metódusnak egy `dictionary` paraméter amely, kulcs-érték párokat kell tartalmazzon. Így a szolgáltatás lekérésekor a következő módon hivatkozhatunk a szükséges szolgáltatásra:

```
serviceReference = context.getServiceReferences(Example.class.getName(), "(Name=ExampleBundle)
    ");
service          = (Example) context.getService(serviceReference[0]);
```

A szolgáltatások egy másik fontos tulajdonsága az, hogy megőrzik az állapotukat. Amennyiben lekérésre kerül egy szolgáltatás egy batyu által, amely használja is a szolgáltatás metódusait, aztán ugyanezen szolgáltatás ismét lekérésre kerül egy másik batyu által, amely szintén szeretné használni a szolgáltatás metódusait, ő már az előző batyu által beállított értékekkel fog találkozni. Ennek a megoldására az OSGi keretrendszer definiál egy `ServiceFactory` interfészt, amely két metódussal rendelkezik:

```
public class ExampleServiceFactory implements ServiceFactory {

    public Object getService(Bundle bundle, ServiceRegistration registration) {
        Example example = new ExampleImpl();
        return example;
    }

    public void ungetService(Bundle bundle, ServiceRegistration registration,Object service) {
    }
}
```

Az `ExampleServiceFactory` osztály mindig egy új példányát adja vissza az `ExampleImpl` szolgáltatásnak. Ahhoz, hogy használható legyen az `ExampleServiceFactory` osztály annyi módosításra van szükség a fenti példához képest, hogy a `context.registerService()` metódus, nem az `ExampleImpl` osztály egy példányát fogja paraméterként megkapni, hanem az `ExampleServiceFactory` osztály egy példányát.

```
serviceRegistration  = context.registerService(Example.class.getName(), new
    ExampleServiceFactory(), dictionary);
```

A szolgáltatások közzététele megtörténhet a batyu indulásakor, illetve futási időben is.

5. fejezet

A rendszer használata

5.1. Teszt indítása

6. fejezet

Következtetés és továbbfejlesztési lehetőségek

A RoboRun projekt keretein belül megvalósításra került egy megerősítéses tanulási algoritmusok tesztelésére szolgáló teljes rendszer, melyhez tartozik egy webes felület, illetve egy adatbázis. A rendszer teljesen az OSGi keretrendszerre épül.

A RoboRun projekt architektúrája tervezésekor óriási hangsúly volt fektetve a továbbfejleszthetőségre, így a projekt felépítése is ezt tükrözi.

Az OSGi keretrendszer által minden komponens külön van választva, így a már meglévő részek rugalmasan továbbfejleszthetőek és kiegészíthetőek. A RoboRun projekt jelen formájában egy OSGi konténerben fut, mely egy távoli elérésű GlassFish szerveren fut.

A projekt továbbfejlesztésére számos lehetőség létezik. Egyik legfontosabb ilyen lehetőség, a projekthez egy Eclipse Plugin[3] készítése, mely által az Eclipse fejlesztői környezet, egy olyan környezetet garantálhat mely megkönnyíti a megerősítéses tanulási algoritmusok implementálását, illetve ez által megvalósítható az, hogy az Experiment program a saját gépünkről fusson, míg a rendszer többi része egy központi elérésű szerveren található. E mellett a webes felület is kiegészíthető számos funkcióval, ami még látványosabbá teheti a tesztek futtatását.

Irodalomjegyzék

[1] Apache felix hivatalos weboldal. http://felix.apache.org/. Utolsó megtekintés dátuma: 2015-05-15.

[2] Paller gábor - osgi és batyuk. http://pallergabor.uw.hu/hu/java-app/OSGi.html. Utolsó megtekintés dátuma: 2015-05-15.

[3] Eclipe plugin hivatalos weboldal. http://marketplace.eclipse.org/. Utolsó megtekintés dátuma: 2015-05-15.

[4] Eclipe equinox hivatalos weboldal. http://eclipse.org/equinox/. Utolsó megtekintés dátuma: 2015-05-15.

[5] Git hivatalos weboldal. https://github.com/. Utolsó megtekintés dátuma: 2015-05-15.

[6] Glassfish hivatalos weboldal. https://glassfish.java.net/. Utolsó megtekintés dátuma: 2015-05-15.

[7] Maven hivatalos weboldal. https://maven.apache.org/. Utolsó megtekintés dátuma: 2015-05-15.

[8] Osgi hivatalos weboldal. https://osgi.org/download/r6/osgi.core-6.0.0.pdf. Utolsó megtekintés dátuma: 2015-05-15.

[9] Redmine hivatalos weboldal. http://www.redmine.org/. Utolsó megtekintés dátuma: 2015-05-15.

[10] Rl- glue hivatalos weboldal. http://glue.rl-community.org/wiki/Main_Page. Utolsó megtekintés dátuma: 2015-05-15.

[11] Szolgáltatás orientált architektúta wikipedia. http://hu.wikipedia.org/wiki/Szolgáltatásorientált_architektúra. Utolsó megtekintés dátuma: 2015-05-15.

[12] Tortoise git. https://code.google.com/p/tortoisegit/. Utolsó megtekintés dátuma: 2015-05-15.