

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

BABEȘ-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 BABEȘ-BOLYAI, CLUJ-NAPOCA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
SPECIALIZAREA INFORMATICĂ

Lucrare de licență

**Platforma de evaluare al algoritmilor de  
instruire prin întăriri, bazată pe OSGI**



CONDUCĂTOR ȘTIINȚIFIC:  
LECTOR DR. JAKAB HUNOR

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

<b>1. Bevezető</b>	<b>3</b>
<b>2. Alapfogalmak</b>	<b>5</b>
2.1. A megerősítéses tanulás . . . . .	5
2.2. A megerősítéses tanulás algoritmusok kipróbálásának alap lépései . . . . .	6
2.2.1. Kihívások a kísérletek futtatása során . . . . .	7
2.3. Alapfogalmak bevezetése . . . . .	7
<b>3. Felhasznált technológiák és eszközök</b>	<b>11</b>
3.1. Alkalmazáserver . . . . .	11
3.2. Webalkalmazás . . . . .	11
3.2.1. A Vaadin architektúra . . . . .	12
3.2.2. A Vaadin komponensek . . . . .	12
3.3. Verziókövetés . . . . .	13
3.4. Projektmenedzsment . . . . .	15
3.5. Build rendszer . . . . .	15
3.6. További felmerülő problémák megoldása . . . . .	16
<b>4. Az OSGi specifikáció</b>	<b>17</b>
4.1. Az OSGi alkalmazás modell . . . . .	17
4.2. Az OSGi architektúra . . . . .	17
4.2.1. OSGi szolgáltatások . . . . .	21
4.3. A RoboRun projekt és az OSGi . . . . .	23
<b>5. A rendszer felépítése és használata</b>	<b>28</b>
5.1. Alapelképzelés . . . . .	28
5.2. A rendszer felépítése . . . . .	28
5.2.1. Webes felület . . . . .	31
5.2.2. Webes felület funkcionalitásai . . . . .	32
5.3. A rendszer telepítése . . . . .	33
5.4. Tesztek futtatásának folyamata . . . . .	34
<b>6. Következtetés és továbbfejlesztési lehetőségek</b>	<b>36</b>

## 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 dolgozatban bemutatásra kerülnek a projekt főbb funkcionálisai 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[9] specifikációra épül, amely egy dinamikus, modularizált komponens modellt definiál komplex alkalmazások felépítésére. E mellett teljesen egységes, hiszen a teljes rendszer saját konvenciók alapján került felépítésre. Fontos megemlíteni, hogy a rendszer könnyen elérhető és használható.

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 felépítése egyedinek számít a piacon, viszont létezik néhány projekt, amely rendelkezik hasonló funkcionálisokkal. A RoboRun projekt szempontjából az RI-Glue<sup>1</sup> projektet érdemes kiemelni, hiszen ez szolgált a RoboRun projekt alapjául és számos funkcionálisa is felhasználásra került a projekt során. Az RI-Glue[13] projekt szerzői Brian Tanner és Adam White. E projekt eredetileg C++ ban íródott, viszont van teljesen Java- ban megírt változata is. Az RI-Glue egy nyelv független környezet a megerősítéses tanulási algoritmusok tanulmányozására. Kétféle protokollt kínál, az úgynevezett 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<sup>2</sup>, ClSquare<sup>3</sup>, PIQLE<sup>4</sup>.

A RoboRun projekt az RI-Glue projektet tovább gondolva és funkcionálisait felhasználva, napjaink

---

1. [http://glue.rl-community.org/wiki/Main\\_Page](http://glue.rl-community.org/wiki/Main_Page)

2. <http://www.igi.tu-graz.ac.at/gerhard/rl-toolbox/general/overview.html>

3. <http://ml.informatik.uni-freiburg.de/research/clsquare>

4. <http://piqle.sourceforge.net/>

## 1. FEJEZET: BEVEZETŐ

technológiáin alapszik. Egy dinamikus és egységes környezetet biztosít, mindezt úgy, hogy a rendszer 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 RI-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 átláthatósága mellett, könnyen kezelhető és folyamatosan elérhető. 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 megtekinthetők.

A dolgozat hat fejezetből áll. Az első fejezet bevezetőként szolgál.

Az második fejezet röviden ismerteti a mesterséges intelligenciát, majd bemutatja a megerősítéses tanulást néhány világbeli példán keresztül. E fejezet kitér a megerősítéses tanulási kísérletek esetén felmerülő problémákra és kihívásokra. Legvégül bevezetésre kerül néhány alapfogalom.

A harmadik fejezet a projekt által felhasznált technológiákat és eszközöket ismerteti. Bemutatja, hogy miért esett ezen technológiákra és eszközökre a választás a tervezés során, illetve a fejezet ismerteti, hogy ezek pontosan milyen szerepet játszottak a projekt megvalósítása során.

A negyedik fejezet a RoboRun projekt alapjául szolgáló OSGi alkalmazás modellt mutatja be, ismerteti ennek architektúráját, illetve azt, hogy miért esett az OSGi-ra a választás. E fejezet kitér arra is, hogy a projekt, hogyan használja az OSGi- t a megerősítéses tanulási kísérletek lebonyolítására.

Az ötödik fejezet a rendszer teljes felépítés mutatja be. Külön kitér a webes felület felépítésének részleteire, illetve bemutatja a rendszer telepítését és használatát.

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. E mellett bemutatásra kerülnek a megerősítéses tanulási algoritmusok alaplépései, illetve a RoboRun projekttel kapcsolatos néhány alap fogalom és ezek használata a projekt során.

### 2.1. A megerősítéses tanulás

Már az 1950 - es évek előtt foglalkoztak mesterséges intelligencia kutatással, csak akkor még nem így nevezték. Az 1950 - es években John McCarthy<sup>1</sup> megalkotja a mesterséges intelligencia kifejezést. Az intelligencia fogalmát Marvin Minsky<sup>2</sup> a következőképpen definiálta: "Az intelligencia egy gyakran használt fogalom annak a rejtélynek a kifejezésére, hogy néhány önálló elem, vagy elemek felelősek a személy következtetési képességéért. Én jobban szeretem úgy elképzelni ezt, mint amely nemcsak valami különös erőt, vagy tüneményt reprezentál, hanem egyszerűen az összes mentális képességet, amelyet mi minden pillanatban megcsodálhatunk, de még nem értettünk meg."

Az ember már nagyon rég próbálkozik azzal, hogy a természettől kapott képességeit, mesterséges eszközökbe beültesse, illetve ezen eszközök segítségével kibővítsse. Ennek megvalósítására a számítógépek megjelenésével nyílt igazán jó lehetőség. A számítógépek lehetőséget biztosítanak arra, hogy az emberi intelligenciát részben helyettesítsék. Rengeteg kutatás és kísérletezés folyik ennek érdekében.

A gépi tanulás is a természetből indul ki. Az élő szervezetet próbálja modellezni. Ezen algoritmusok legfőbb jellemzője az adaptációs<sup>3</sup> tanulási képesség. A tanulás és az adaptáció valójában az élő szervezet működését jellemzi, ahogyan az ember is megszerzett ismeretek és tapasztalatok alapján cselekszik. Tanulásról beszélünk abban az esetben is, ha tapasztalatok sorozata kerül megtanulásra, illetve abban az esetben is amennyiben előző tapasztalatok alapján képes különböző döntések meghozatalára.

Ahogyan az élő szervezeteknél, úgy a gépeknél is többfajta tanulásról beszélhetünk. Például a neurális hálók<sup>4</sup> esetén minták alapján történik a tanulás. Ez azt jelenti, hogy nagy mennyiségű adatból próbálunk megfelelő mennyiségű ismeretet szerezni és ezáltal befolyásolni a rendszer működését. A rendszer működésének befolyásolása több dologra is irányulhat. Például, hogy a rendszer adott bemenetekre, előre megadott válaszokat produkál- e. Lehet olyan eset is, amikor azt szeretnénk tesztelni, hogy a rendszer képes- e adott bemenetekre valamilyen szabályosságot felállítani. Ezen algoritmusokat gyakran helyezik változó környezetekbe és azt tesztelik, hogy mennyire képesek alkalmazkodni az új környezethez.

1. [http://en.wikipedia.org/wiki/John\\_McCarthy\\_%28computer\\_scientist%29](http://en.wikipedia.org/wiki/John_McCarthy_%28computer_scientist%29)

2. [http://en.wikipedia.org/wiki/Marvin\\_Minsky](http://en.wikipedia.org/wiki/Marvin_Minsky)

3. <http://hu.wikipedia.org/wiki/Adaptáció>

4. [http://hu.wikipedia.org/wiki/Neurális\\_hálózat](http://hu.wikipedia.org/wiki/Neurális_hálózat)



A mesterséges intelligencia esetén beszélhetünk felügyelt, nem felügyelt és félig felügyelt tanulási módszerekről. A megerősítéses tanulás a nem felügyelt tanulási ágat képviseli. A nem felügyelt tanulás esetén nem állnak rendelkezésre adott bemenetekhez tartozó elvárt válaszok, tehát a rendszer nem rendelkezik a tanító adatok halmazával. A rendszernek a bemenetek és kimenetek alapján kell valamilyen viselkedést kialakítania. A környezettől nem kap semmiféle visszajelzést annak érdekében, hogy a hálózat jól vagy rosszul viselkedik-e. Ezáltal megállapítható, hogy ezen algoritmusok legfőbb jelmezője az, hogy nem előre meghatározott képességekkel rendelkeznek, amelyek csak egy adott feladat elvégzésére elegendőek, hanem képesek arra, hogy folyamatosan fejlesszék képességeiket és ezáltal képesek legyenek alkalmazkodni új és ismeretlen környezetekhez.

### 2.2. A megerősítéses tanulás algoritmusok kipróbálásának alap lépései

A megerősítéses tanulási[12] algoritmusok állapotmegfigyeléseken és jutalmakon alapulnak. Ez szintén az élő szervezetekre vezethető vissza, hiszen az állatok tapasztalatainak megszerzése is az idegrendszer állapotmegfigyelésén alapszik. Megfigyelhető például a kutyák esetében, ha egy forró tárgyhoz hozzá ér hamar elkapja a mancsát, viszont még néhányszor próbálkozik. Majd néhány próbálkozás után megtanulja, hogy egy adott tárgy ha forró, ahhoz többet ne érjen hozzá. Minden egyes algoritmus bizonyos számú epizódot hajt végre, mely rendelkezik bizonyos számú lépés sorozattal. Az algoritmusok futtatásának a célja, hogy egy optimális stratégiát alakítsanak ki a feladat megoldására illetve, hogy maximalizálják a jutalmakat. A jutalmak maximalizálása egyben a feladat elvégzésének a költségét minimalizálja.[3]

A megerősítéses tanulási algoritmusok általában egy előre definiált állapotból indulnak, az éppen aktuális problémának megfelelően. Mivel nincs semmilyen információjuk arról, hogy mi lenne a jó lépés így véletlen lépésekkel próbálkoznak. Minden egyes lépés után kiértékelik a kialakult állapotot, ezt nevezzük állapotmegfigyelésnek. Viszont a rendszernek nincs semmilyen tudomása arról, hogy a kialakult állapot jó vagy rossz. Így az algoritmusnak semmilyen alapja nem lesz arra, hogy milyen lépést kellene hozzon. Tehát az algoritmusnak szüksége van arra, hogy tudja, ha valami jó történt, illetve ha valami rossz. E miatt az egyes állapotokhoz különböző jutalmakat rendelnek. A jutalmak adása kezdetben valamilyen becslés alapján történik a jövőre nézve, ezt nevezik a jutalmak hosszútávú maximalizációjának. Egyes környezetekben a jutalmak csak a teszt végén jelennek meg, például a sakk esetén, míg más környezetben folyamatosan jönnek a jutalmak, például a pingpong esetén minden pont jutalomnak tekinthető.

A megerősítéses tanulási algoritmusok esetében három fő részt lehet elkülöníteni. Az *Agent*, azaz az Ügynök, ami valójában a tanulási algoritmus. Az *Environment*, azaz a Környezet, amely meghatározza az adott tesztet, például a sakk problémája. Az *Experiment*, azaz a Kísérlet, amely meghatározza az epizódok számát, illetve az epizódokon belül a lépések számát. E három fő részről és az ezek közötti kommunikáció részletesebb leírását az 2.3 szekció tartalmazza.

### 2.2.1. Kihívások a kísérletek futtatása során

A megerősítéssel tanuló algoritmusok futtatása saját számítógépen nem a legjobb megoldás. Ezen kísérletek futtatása egy időigényes folyamat lehet, például egy robot esetén, melynek teljes mozgását reprezentálni szeretnénk. A robot összes mozgásának reprezentálása nagyon sok lépést vehet igénybe, annak érdekében, hogy sikeresen kialakításra kerüljön az optimális mozgási stratégia. A kísérletek futtatása során felmerül az a probléma, hogy egy adott kísérlet több példányát kellene végrehajtani egyidejűleg, illetve több különböző kísérlet példányait egyszerre mindezt online. Amennyiben egy kísérlet futási ideje több óra, nap esetleg hét is lehet, a fejlesztők nem várhatnak egy adott kísérlet befejezésére, hiszen akkor az előre haladás nagyon nagyon lassú lenne. Az időigényesség mellett fontos megemlíteni, hogy komplexebb szimulációs környezetek esetén, melyet egy komplex tanuló algoritmus irányít, nagyon nagy erőforrás igényre lehet szükség. Amennyiben már egy algoritmus végrehajtása nagy erőforrás igényű lehet, akkor több algoritmus egyidejű végrehajtása esetén óriási erőforrásigényekről beszélhetünk. A kísérletek nagy mennyiségű adatokat generálnak, melyekre a későbbi kiértékelés és esetleges összehasonlítások során szükség lehet. Ezen adatok tárolására, adatbázisokra lehet szükség a könnyed hozzáférés érdekében. Ezen kihívások azt eredményezik, hogy a számítógép állandóan be kell legyen kapcsolva, a nagy erőforrás igény miatt másra nem is lehetne használni, illetve nehezen kezelhetőek lennének a tesztek.

Ennek megoldására egy olyan szimulációs környezet megvalósítása kínál lehetőséget, amely egy távoli, jól felszerelt szervergépen fut, mely folyamatosan elérhető az interneten keresztül és rendelkezik a megfelelő erőforrásokkal. E környezet dinamikusan kell működjön, hiszen egyszerre több teszt futtatását kell lehetővé tegye. E mellett rendelkeznie kell valamilyen adatbázis kapcsolattal, ahova minden teszt esetén elmenti az adatokat. Ezen tulajdonságok mellett, elengedhetetlen a szimulációs környezet számára a könnyed bővíthetőség és módosíthatóság tulajdonsága, amely által a már meglévő kísérletek könnyedén megváltoztathatóak az új elképzelések alapján, illetve az új kísérletek hozzáadása könnyedén eszközölhető.

## 2.3. Alapfogalmak bevezetése

A projekt megvalósítása során a szerző megismerte az RL-Glue projekt elveit és annak funkcionalitásait, melyek meghatározó szerepet töltenek be a RoboRun projektben is. A három alap komponens mindkét projekt esetén az *Agent*(Ügynök), *Environment*(Környezet) és az *Experiment*(Kísérlet). Ezen komponensek egymással való interakciója révén nyílik lehetőségünk futtatni illetve tesztelni a megerősítéssel tanuló algoritmusokat.

Az *Agent* komponens valójában a tanuló 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

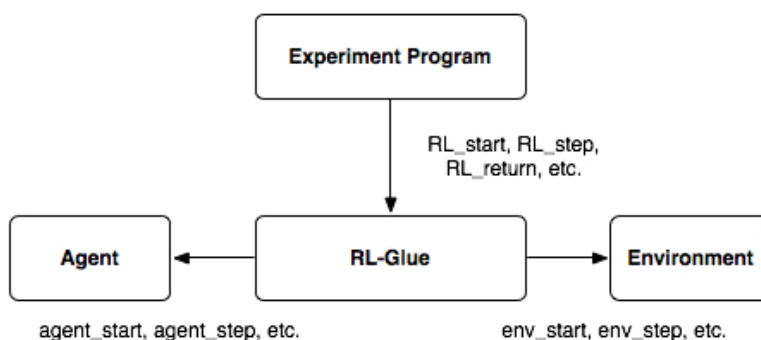
## 2. FEJEZET: ALAPFOGALMAK

kell próbálnia, majd az ezekből megszerzett tudást, ami esetünkben a jutalom, optimális módon felhasználja a következő cselekvés meghatározására. Például `RandomAgent`, mely valójában nem is tanulási algoritmus, hiszen véletlenszerű lépések alapján próbál az egyes környezetekben megoldást találni a problémára. Viszont gyakran használják egyes környezetek kipróbálására.

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 lépés sorozat után, amennyiben az intervallum alsó határához közelít a szám, negatív a visszajelzés. Például a `MountainCar` környezet, amelyben az a feladata a tanuló algoritmusnak, hogy egy bizonyos kezdeti pozícióból eljusson egy kijelölt végső pozícióba a lehető leggyorsabban. Ehhez meg kell tanulnia a szakadékból előre és hátra mennie egészen addig, amíg olyan sebességre tesz szert, hogy elegendő a végpontba való eljutáshoz.

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. Az RL-Glue projekt alap kommunikációját az `Agent`, `Environment`, `Experiment` és RL-Glue között a 2.1 ábra szemlélteti.

How RL-Glue Interacts with the Experiment Program, Agent and Environment



2.1. ábra. RL-Glue projekt komponensek közti kommunikációja:

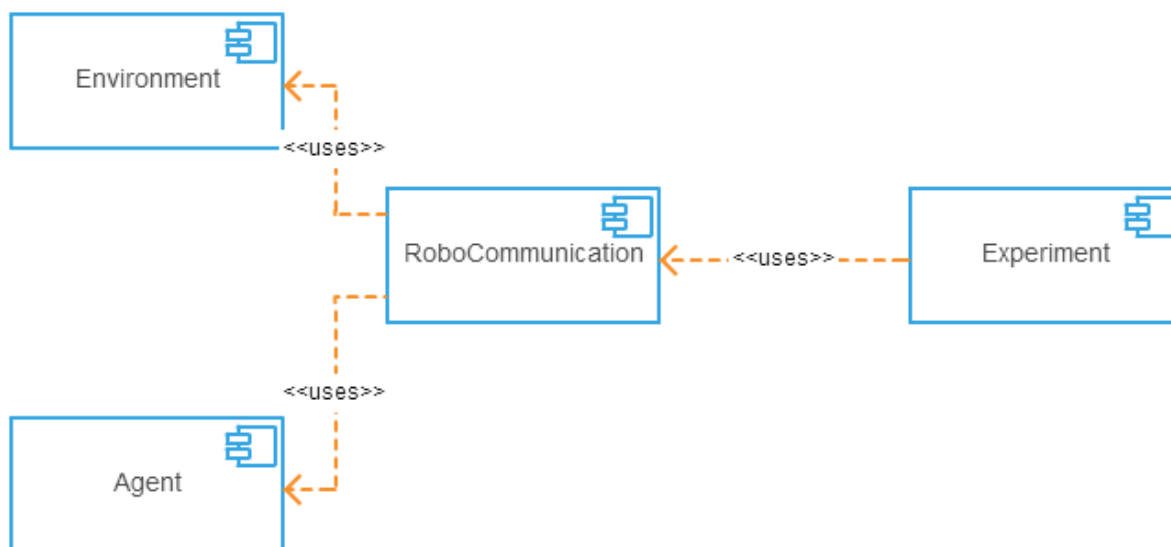
<http://rl-glue.googlecode.com/svn/trunk/docs/html/index.html>

## 2. FEJEZET: ALAPFOGALMAK

A RoboRun projekt esetén az RL-Glue komponens helyét felváltja a RoboCommunication komponens, a funkcionalitásokat tekintve az alap ötlet megmaradt viszont számos új dologgal ki lett bővítve, illetve új funkcionalitások kerülnek használatra. Számos funkcionalitás található az RL-Glue projektben, amelyre a RoboRun projektben nincs szükség, így ezen funkcionalitások teljesen ki lettek hagyva. Ilyen például a hálózat alapú kommunikáció.

Az Agent, Environment, Experiment és a RoboCommunication komponensek interakciójának sorrendje hasonlóan működik, mint az RL - Glue projektben megvalósított elgondolás, viszont a RoboRun projekt esetén minden egyes komponens egy szolgáltatás az OSGi konténerben, amelyek képesek egymással kapcsolatba lépni az OSGi szolgáltatásokon keresztül, így megvalósítva a dinamikus, komponens alapú modell kivitelezését és a több teszt egy időben való futtatási lehetőségét. E mellett ezen szolgáltatások elérése korlátozott, a többi szolgáltatás számára, amely biztonsági okokból is előnyös a rendszerre nézve. Az OSGi konténer futtatásáért egy GlassFish[7] szerver a felelős. Az OSGi specifikáció részletes ismertetése a 4 fejezetben olvasható.

A konténerben egyszerre több előre definiált Agent és Environment lehet telepítve, ezek száma nincs korlátozva, annyi telepíthető belőlük amennyi még nem okoz gondot a szerver futtató számítógép hardver konfigurációjának. Bármikor módosítható, törölhető vagy teljesen új Agent és Environment is hozzáadható a konténerhez anélkül, hogy a szervert meg kellene állítani vagy újra kellene indítani. A RoboCommunication komponensből egyet tartalmaz a rendszer, hiszen ez az egy szolgáltatás képes kielégíteni számtalan Agent és Environment komponens példányt egy időben, mindezt a projekt architektúrájának és az OSGi keretrendszerben rejlő lehetőségek által. Az Experimentekből is egyszerre több lehet telepítve, hiszen ezek felelnek egy egy teszt indításáért. Egyszerre több tesztet is lehet indítani, vagy lehetőség van arra is, hogy különböző időközönként telepítsünk egy- egy Experiment komponenset. A RoboRun projekt alap komponensei közötti kommunikációt szemlélteti a 2.2 ábra.

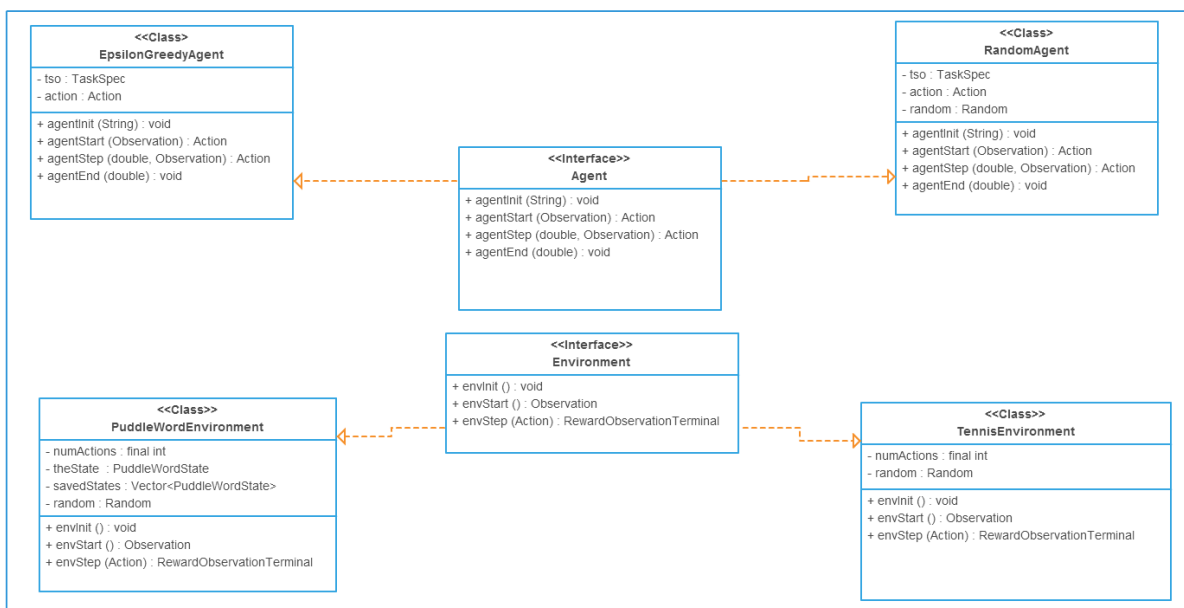


2.2. ábra. A RoboRun projekt alap komponensek közti kommunikáció:

## 2. FEJEZET: ALAPFOGALMAK

Egy kísérlet futtatásához szükségünk van egy Agent, egy Environment, egy Experiment és egy RoboCommunication komponensre. A RoboCommunication komponens esetén még szükség van néhány függőségre az adatbázishoz való hozzáférés biztosítására, illetve a webes felület adatainak feltöltése érdekében, ezekről részletesebben 5 fejezetben lehet megismerkedni. A kísérlet belépési pontjaként az Experiment komponens szolgál, hiszen ő határozza meg, hogy melyik Agent, illetve Environment példánnyal szeretne dolgozni, illetve az Experiment határozza meg a szükséges lépések számát iterációnként és az iterációk számát. Az Experiment komponens lekérdezi a szükséges szolgáltatásokat és átadja a RoboCommunication szolgáltatásnak a lekérdezett Agent és Environment szolgáltatás példányokat. A RoboCommunication komponens fogja biztosítani a három komponens között a folyamatos kommunikációt a teszt futtatása során. A tesztek állapotának nyomon követésére lehetőség van a webes felületen keresztül, illetve a tesztek befejeztével megtekinthetőek a futása során létrejött adatok.

A projekt architektúrája követi a OSGi szabványokat, így minden szolgáltatásnak implementálnia kell egy interfészt, mely egy külön batyuban található. Az interfész közzéteszi a szolgáltatások számára a csomagját. A csomag importálása által a szolgáltatások megvalósíthatják az adott interfészt. Minden egyes Agent kötelező módon implementálja az AgentInterface -t és minden Environment kötelező módon implementálja az EnvironmentInterface -t. Ez által valósul meg a szimulációs környezet egységessége, amely egy egyedi konvencióra épül. Nem létezhet olyan Agent vagy Environment példány a rendszerben, amely nem implementálja a neki megfelelő interfészt. Amennyiben mégis telepítésre kerül egy ilyen szolgáltatás, nem fog működni, mert egy szolgáltatás lekérdezése csak az általa implementált interfész által lehetséges. Az interfészek és a hozzájuk tartozó implementációkat a 2.3 ábra szemléltet.



2.3. ábra. Agent és Environment Interfész és implementáció

### 3. fejezet

## Felhasznált technológiák és eszközök

**Összefoglaló:** A fejezet ismerteti a RoboRun projekt által felhasznált alkalmazás szervert és ennek szerepét, illetve a webalkalmazáshoz felhasznált technológiát, mely a projekt webes felületének megvalósításában játszott szerepet. A fejezet célja, hogy bemutassa azon eszközöket, amelyek meghatározó szerepet töltek be a projekt megalkotása során.

### 3.1. Alkalmazáserver

A RoboRun projekt megvalósításához elengedhetetlen egy alkalmazáserver használata, hiszen az OSGi konténernek szüksége van egy alkalmazáserverre, amely képes futtatni és ez által folyamatosan elérhetővé tenni az alkalmazást az interneten keresztül. Bővebben az OSGi specifikációról a 4 fejezetben lehet olvasni.

Az alkalmazáserver nem más mint egy szoftver keretrendszer, mely lehetőséget biztosít tetszőleges alkalmazások futtatására. A Java Enterprise Edition specifikáció több referencia implementációval rendelkezik. Ilyen referencia implementációk például: JBoss As<sup>1</sup>, GlassFish<sup>2</sup>, WebSphere<sup>3</sup>, WebLogic<sup>4</sup>.

A RoboRun projekt tervezésekor ezen alkalmazászerverek közül a Glassfish referencia implementációja tűnt megfelelőnek, mely az Oracle tulajdonába tartozik és teljesen nyílt forráskódú, viszont vásárolható hozzá kereskedelmi licenz támogatás, amelyben az Oracle saját megoldásai is helyet kapnak, például képes teljes domainek mentésére és visszaállítására. A Glassfish fontos tulajdonsága, hogy képes OSGi konténerek kezelésére. A RoboRun projekt esetén a Glassfish alkalmazáserverre telepített OSGi konténerben vannak elhelyezve a különálló komponensek, amelyek együtt alkotják a RoboRun projektet. A Glassfish alkalmazáserver révén távolról is telepíthetők könnyedén új komponensek, illetve használhatóak. A Glassfish manageli a RoboRun projekt webes felületét is.

### 3.2. Webalkalmazás

A projekt webes felületének megvalósítására rengeteg módszer, eszköz és keretrendszer áll rendelkezésre, melyek közül lehet választani. A RoboRun projekt tervezésekor fontos szempont volt a könnyed és gyors használhatóság, illetve a dinamikus módosíthatóság tulajdonsága. Ezek alapján a Vaadin[16] webalkalmazás-keretrendszere esett a választás.

---

1. <http://jbossas.jboss.org/>

2. <https://glassfish.java.net/>

3. <http://www.ibm.com/software/websphere>

4. <http://www.oracle.com/technetwork/middleware/weblogic/overview/index-085209.html>

### 3. FEJEZET: FELHASZNÁLT TECHNOLOGIÁK ÉS ESZKÖZÖK

A Vaadin egy olyan Java webalkalmazás-keretrendszer, amely lehetőséget biztosít gazdag webalkalmazások<sup>5</sup> fejlesztésére. A Vaadin lehetőséget nyújt a felület Java nyelvben történő implementálására, illetve egy AJAX alapú kommunikációs modellt biztosít. A Vaadin architektúra két fő részből tevődik össze: a kliens oldali részből, amely tartalmazza a Google Web Toolkit -et<sup>6</sup> és amely a Java kódot, JavaScript kódra fordítja, illetve a szerver oldali részből, amely JavaServlet technológiát használ. A szerver oldali rész tartalmazza a felhasználó felület létrehozásához szükséges komponenseket. Ezen komponensek nagyon hasonlítanak a Java-ban írt standard alkalmazásoknál használt AWT, illetve SWING komponensekre. A Vaadin-os komponensek is figyelőket és eseményeket használnak. Az alapértelmezett komponens és téma készlet mellett, telepíthetők különböző kiegészítők a még könnyebb használat és a még látványosabb felhasználói élmény érdekében. A komponensek személyre szabhatóak a CSS, HTML5, JavaScript technológiák felhasználása által.

#### 3.2.1. A Vaadin architektúra

A Vaadin webalkalmazás-keretrendszer architektúrája két fő részre osztható fel. A kliens oldal által valószínűleg a megjelenítés, amely JavaScript formájában jelenik meg a böngészőben. A szerver oldali rész mely biztosítja a komponensek könnyed elérését és használatát.

A 3.1 ábra szemlélteti a Vaadin keretrendszer architektúráját. A kliens oldal áll az architektúra legfelső részén, hiszen a kliens közvetlen ezzel kerül kapcsolatba. Ez a réteg a belépési pont ahonnan adatok érkeznek, melyeket a szerver oldali rész fele kell közvetíteni. A kliens oldali részben megtalálható a kliens oldali felhasználói felület és az ehhez tartozó widgetek listája, amelyek a megjelenítésért felelősek. A szerver felelős az üzleti logika megvalósításáért. A Service réteg valósítja meg a kommunikációt a Back-end réteg és a kliens réteg között.

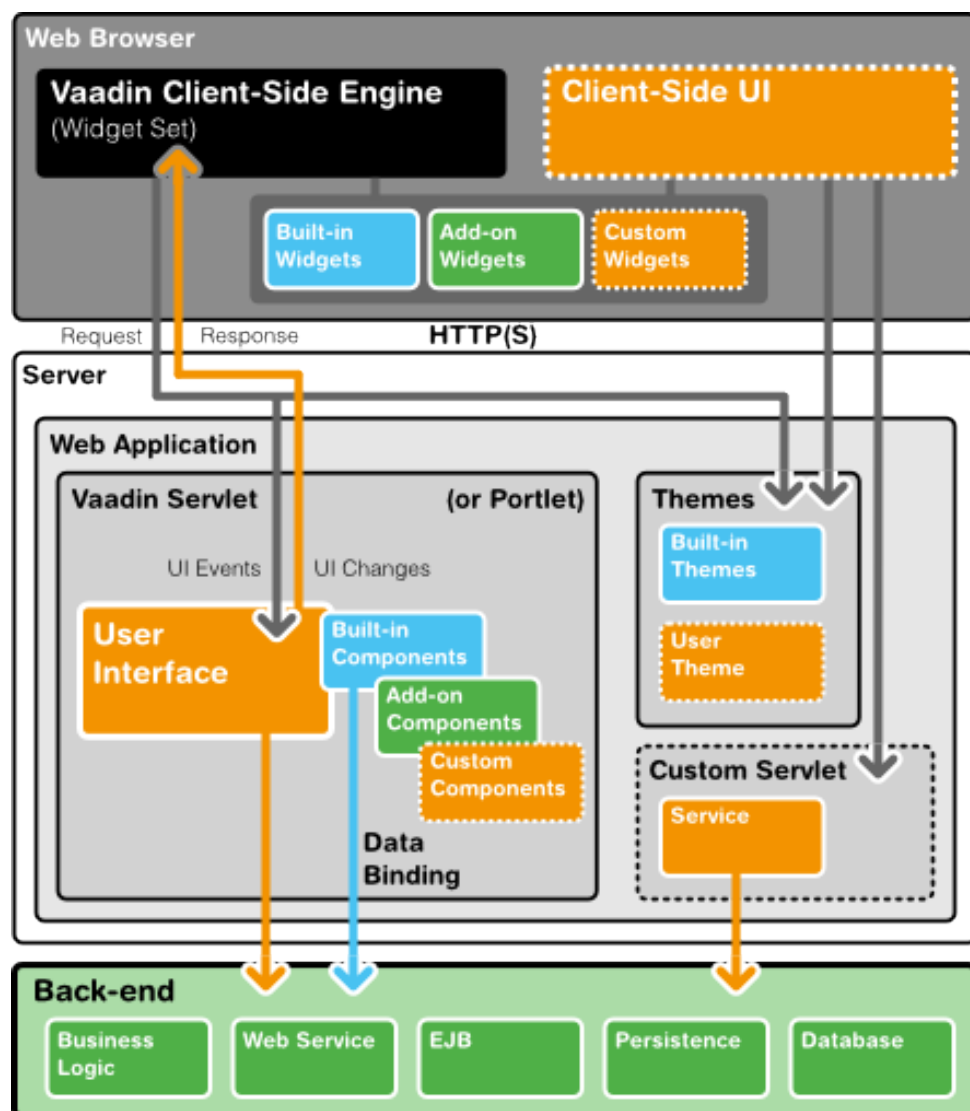
#### 3.2.2. A Vaadin komponensek

A Vaadin webalkalmazás-keretrendszer egy előre definiált komponens gyűjteményt biztosít a fejlesztők számára, a könnyebb munka érdekében. Ezen komponensek használata egyszerű, hiszen nagyon hasonlítanak az AWT és a SWING - es komponensekhez. Az alap gyűjteményben található komponensek által felépíthető egy web alkalmazás teljes felhasználói felülete. Ezen komponensek bővíthetők, teljesen személyre szabhatóak a CSS, HTML5, JavaScript technológiák által. A Vaadin keretrendszer lehetőséget biztosít teljesen új komponensek definiálására is. A Vaadin keretrendszer alap komponensei közötti kapcsolatot, illetve összefüggéseket a 3.2 ábra szemlélteti.

A legfelső réteg a Component interfész, melyet az AbstractComponent absztrakt osztály implementál. Az AbstractComponent közös tulajdonságokkal látja el az őt származtató komponenseket. Az AbstractComponent osztályból származtatva van néhány egyszerű komponens, például a Label(Címke). A Component interfész mellett, megtalálható a Field interfész is, amely öröklí a Component interfészt. Az AbstractField absztrakt osztály

5. [http://hu.wikipedia.org/wiki/Rich\\_Internet\\_Application](http://hu.wikipedia.org/wiki/Rich_Internet_Application)

6. [http://hu.wikipedia.org/wiki/Google\\_Web\\_Toolkit](http://hu.wikipedia.org/wiki/Google_Web_Toolkit)



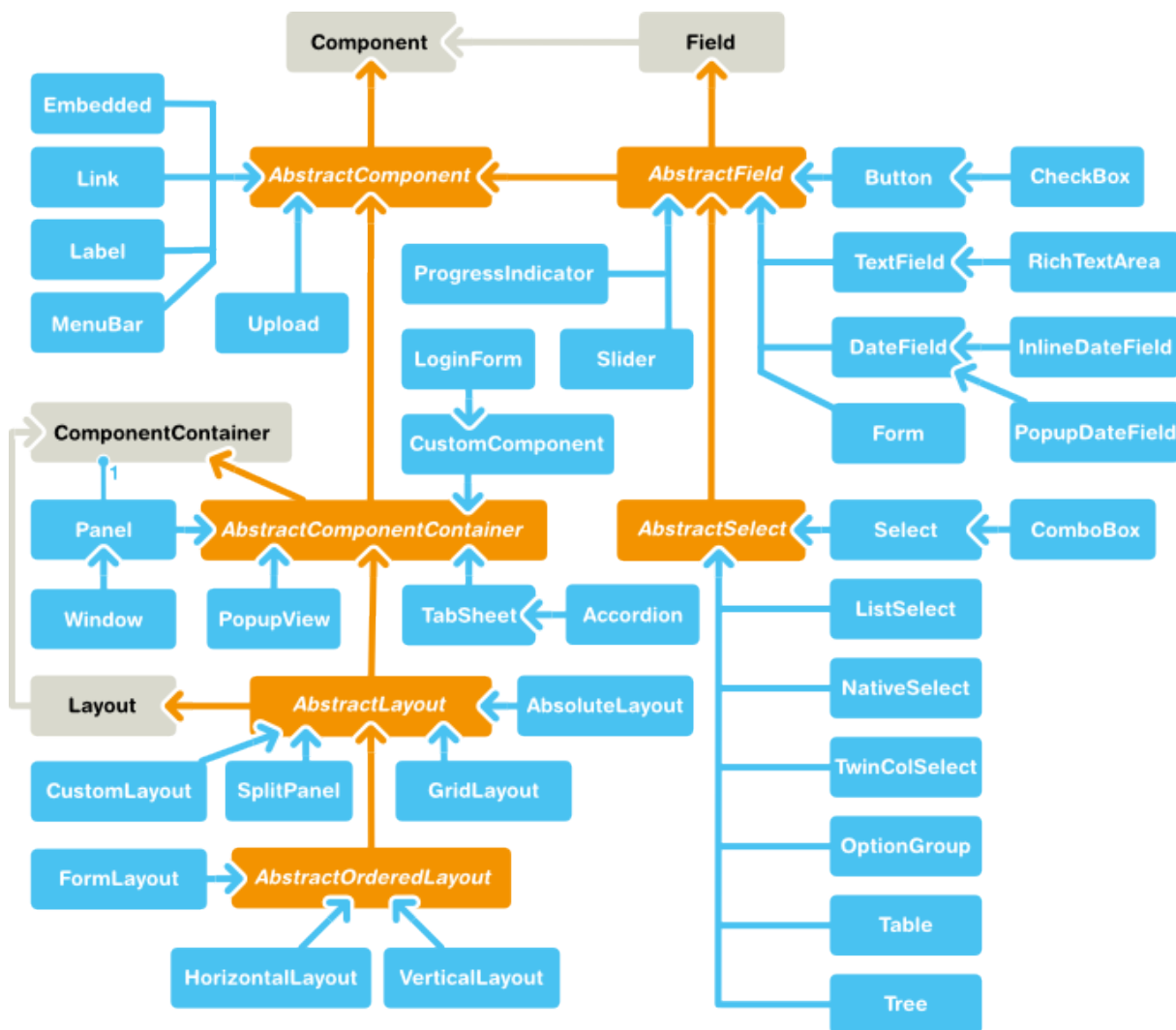
3.1. ábra. Vaadin architektúra <https://vaadin.com/book/-/page/architecture.html>

implementálja a `Field` interfész és örökli az `AbstractComponent` osztály tulajdonságait. Az `AbstractField` osztály a kijelöléssel, navigálással kapcsolatos komponensek alapjait képezi, például `Button` (Gomb). Az `AbstractComponent` osztály tulajdonságait örökli, az `AbstractComponentContainer` osztály is. Az `AbstractComponentContainer` osztályból származnak a konténer - alapú komponensek, például a `Panel`, illetve az elrendezést elősegítő komponensek, például `VerticalLayout` (Függőleges elrendezés).

### 3.3. 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.





3.2. ábra. Vaadin komponensek

<https://inftec.atlassian.net/wiki/display/TEC/Vaadin>

Ezen technológiák nélkül szinte elképzelhetetlen a szoftverfejlesztés, úgy csoportos környezetben mint egyénileg.

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 projekteknek 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 egyszerűen követhető, hogy melyik fejlesztő milyen fázisban tart.

A RoboRun projekt fejlesztése során a forrásállományok tárolására és a fejlesztés nyomkövetésére

### 3. FEJEZET: FELHASZNÁLT TECHNOLOGIÁK ÉS ESZKÖZÖK

felhasznált verziókövető rendszer a Git[6], amely nyílt forráskódú és teljesen ingyenes. Könnyedén megoszthatóak a forrásállományok. A projekt szerzője által használt kliensalkalmazás a TortoiseGit[15]. 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.

#### 3.4. 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."[10]

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, ezáltal megkönnyítve a fejlesztők munkáját.

A RoboRun projekt fejlesztése során alkalmazott projektmenedzsment eszközként a Redmine[11] webes menedzsment eszköz szolgált. A Redmine egy teljesen nyílt forráskódú és platform független projektmenedzsment rendszer. Egyszerű és letisztult felülete révén könnyen kezelhető. Rengeteg funkcionalitást 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.

#### 3.5. Build rendszer

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

A RoboRun projekt fejlesztése során alkalmazott build rendszer a Maven[8], 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. Tehát 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.

### 3. FEJEZET: FELHASZNÁLT TECHNOLOGIÁK ÉS ESZKÖZÖK

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.

#### 3.6. További felmerülő problémák megoldása

A RoboRun projekt esetén szükséges a tesztek futtatásakor az egyes hibaüzenetek megtekintésének lehetősége, melyet a naplózást biztosít. Naplózáshoz az SLF4J<sup>7</sup> - LOG4J<sup>8</sup> párosítás került használatra. Az SLF4J több naplózási keretrendszer fölött képez absztrakciós szintet, így több különböző naplózási implementációt vehetünk igényben általa. A LOG4J implementáció az Apache licenc áll. Az SLF4J naplózási keretrendszernek a legfőbb előnye, hogy a LOG4J bármikor könnyen lecserélhető, bármilyen más implementációra. A projekt esetén adatbázis menedzsment rendszerként a MySQL<sup>9</sup> szolgált, amely többfelhasználós és többszálú SQL - alapú adatbázis-kezelő rendszer. Egy másik felmerülő probléma, az RMI(Remote Method Invocation)<sup>10</sup> problémája. Az eredeti elképzelés szerint a kliens RMI - n keresztül csatlakozott volna a szerverhez, elkérve onnan a telepített szolgáltatások listáját. A listából kiválasztva a neki megfelelő szolgáltatásokat, indított volna teszteket. Tehát az alkalmazás Experiment része teljesen az OSGi konténeren kívül lett volna elérhető. Ennek megvalósítása nem lehetséges, a miatt, hogy az RMI kapcsolaton keresztül elküldött objektumok szerializálásra kerülnek. Az OSGi szolgáltatásokat pedig nem lehet szerializálni, tehát nem elérhetőek a konténeren kívül. Erről részletesebb leírás a 6 fejezetben található.

---

7. <http://www.slf4j.org/>

8. <http://logging.apache.org/log4j/2.x/>

9. <https://www.mysql.com/>

10 [http://en.wikipedia.org/wiki/Java\\_remote\\_method\\_invocation](http://en.wikipedia.org/wiki/Java_remote_method_invocation)

## 4. fejezet

# Az OSGi specifikáció

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

### 4.1. Az OSGi alkalmazás modell

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 specifikáció alapötlete a szolgáltatás orientált architektúrára[14] 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.

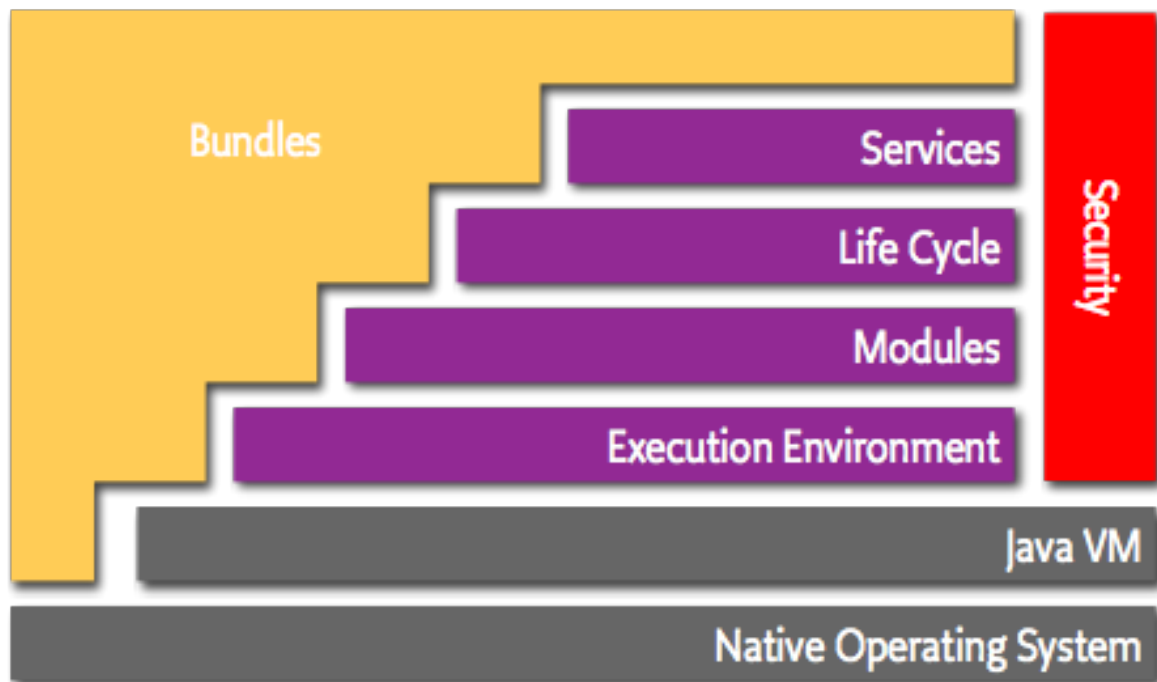
Az OSGi specifikáció egy olyan specifikáció, mely a Java nyelv fölött fut. Az OSGi jelentése, Open Service Gateway Initiative. E specifikáció célja bővíthető Java alkalmazások fejlesztésének a támogatása. Teljesen dinamikus környezetet biztosít, hiszen képes kezelni a csomagok futás idejű megjelenését 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 specifikációnak, például az Apache Felix[1] vagy az Eclipse keretrendszer alapjául szolgáló Eclipse Equinox[5]. A RoboRun projekt esetén az Apache Felix keretrendszer került felhasználásra.

### 4.2. Az OSGi architektúra

Az OSGi különböző eszközöket biztosít a szolgáltatások létrehozása érdekében. Ilyen alap eszközök például a batyuk[2]. Az OSGi architektúrát a 4.1 ábra szemlélteti.

Bundles(batyuk vagy kötegek)

A batyukat az OSGi alapjának tekinthetjük. Általánosan három részből tevődnek össze: Java - kód, statikus erőforrások(pl.: képek) illetve leíró állomány vagy MANIFEST.MF - fájl. A MANIFEST.MF állományokra minden batyunak szüksége van, hiszen ez írja le a futtató környezet számára, hogy az aktuális



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

batyut, hogyan kell használni. Szerepelhet benne az `Activator` osztály neve vagy classpath bejegyzéseket tartalmazhat. A programegységek az OSGi-ban batyuként kerülnek telepítésre. A batyuk rendelkeznek néhány 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 szolgáltatásokat regisztrálni, amelyek által más batyuk számára elérhetővé válnak.

A leíró állomány által értelmezhető a batyu tartalma, melyet a 4.1 kódrészlet szemléltet

Listing 4.1. MANIFEST.MF tartalma

```
1 Bundle-Name: environment
2 Bundle-SymbolicName: environment
3 Bundle-Version: 1.0.0-RELEASE
4 Built-By: Gáll
5 Bundle-Activator: edu.bbte.environment.EnvironmentActivator
6 Created-By: Apache Maven Bundle Plugin
7 Export-Package: edu.bbte.environment
8 Import-Package: edu.bbte.packages.types;version="[0.0,1)", javax.xml.parsers, org.osgi.framework;version="[1.6,2)", org.w3c.dom
```

Az 1-es sortól(**Bundle-Name**) a 4-es sorig(**Bundle-Vendor**) a batyuról tárolt információk találhatók, az 5-ös sor(**Bundle-Activator**) azt az osztályt tartalmazza, amelyik elindul a batyu telepítésekor és annak törlésekor leáll. A 7-edik sor(**Export-Package**) a batyu által közzétett csomagokat tartalmazza, míg a 8-adik sor(**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 mind. Például az 5-ödik sorban (**Bundle-Activator**) található címkét nem

#### 4. FEJEZET: AZ OSGI SPECIFIKÁCIÓ

kötelező megadni, hiszen nem minden batyunak van szüksége `Activator` osztályra.

Az `Activator` osztály vázlatát a 4.2 kódrészlet szemlélteti.

Listing 4.2. Egy `Activator` osztály vázlata

```
1 public class Activator implements BundleActivator {  
2  
3     public void start(BundleContext context) throws Exception {  
4  
5         // ....  
6     }  
7  
8     public void stop(BundleContext context) throws Exception {  
9  
10        // ....  
11    }  
12 }
```

Minden `Activator` osztálynak kötelező módon implementálnia kell a `BundleActivator` interfészt, mely két metódussal rendelkezik, a `start()` és `stop()` metódusokkal. Ezen metódusokat a 3- as illetve a 8- as sor szemlélteti, a 4.2 kódrészletben.

A batyu telepítésekor az OSGi példányosítja az `Activator` osztályt és meghívja a `start()` metódusát automatikusan. A `start()` metódus megkap egy `BundleContext`-re mutató referenciát mely által új szolgáltatásokat 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 a szükséges függőséget csak a teljes batyu képes kielégíteni.

Egy batyuban négyféle csomag érhető el:

A batyu által létrehozott csomagok

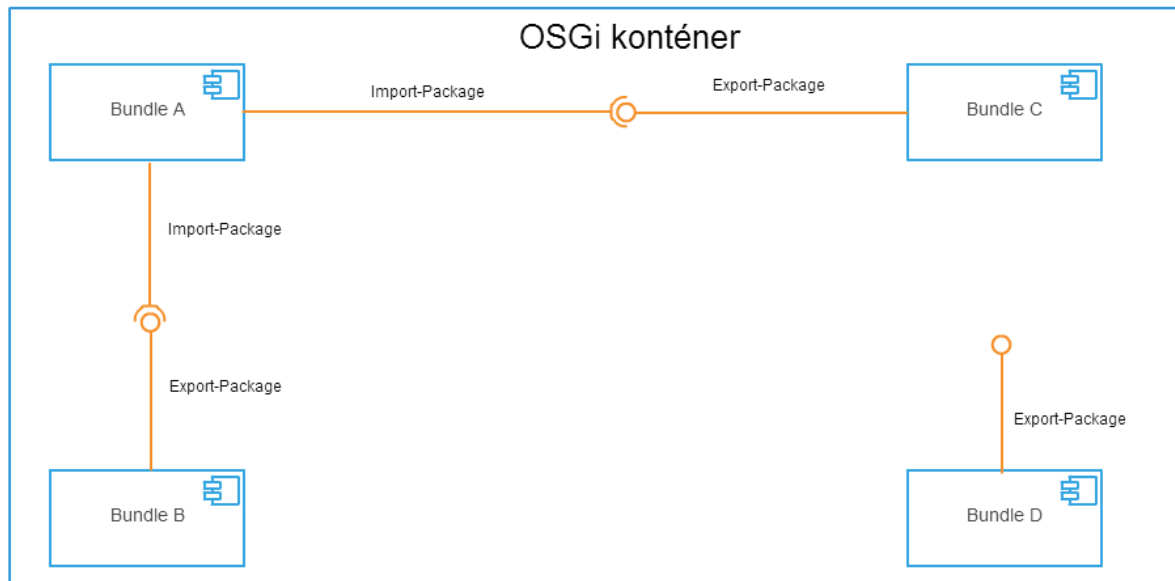
Az **Import-Package** által megadott csomagok

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

A Java összes függvénykönyvtára

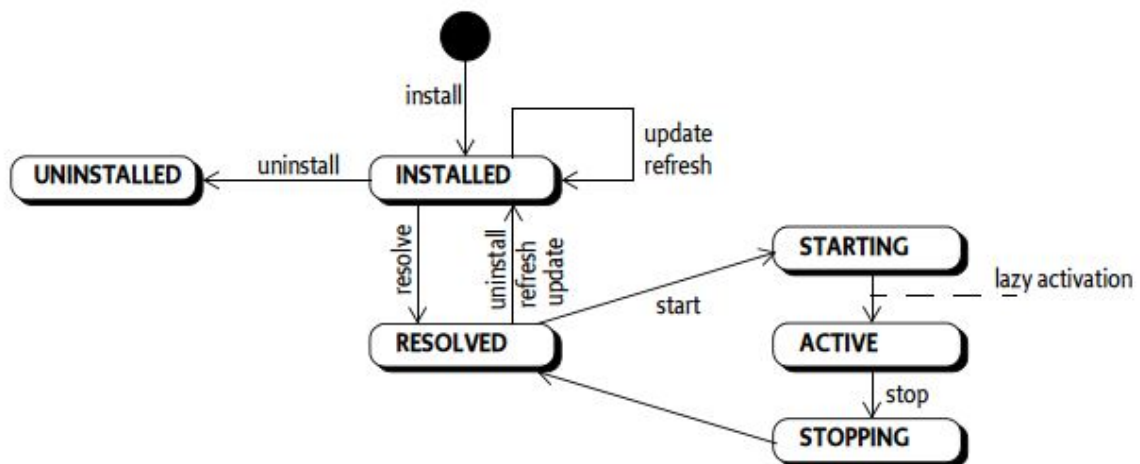
A csomagok Export - Import mechanizmusát a 4.2 ábra szemlélteti.

#### 4. FEJEZET: AZ OSGi SPECIFIKÁCIÓ



4.2. ábra. Batyu által elérhető csomagok

A batyuknak vannak különböző állapotaik, mely által meghatározható, hogy éppen mi történik velük. Ezen állapotok végig kísérik a batyut a telepítés pillanatától egészen a törlésükig. Ezen állapotokat a batyu életciklusának is szokták nevezni. A batyuk életciklusait a 4.3 ábra szemlélteti.



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

**Installed** 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()` módszere 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, futásidőbeli 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 nevéen 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.

Listing 4.3. Szolgáltatás regisztrálása az Activator metódusban tartalma

```
1 public class Activator implements BundleActivator {
2
3     private ServiceRegistration serviceRegistration;
4
5     public void start(BundleContext context) throws Exception {
6
7         serviceRegistration = context.registerService(Example.class.getName(), new
8             ExampleImpl(), null);
9     }
10
11     public void stop(BundleContext context) throws Exception {
12
13         environmentServiceReg.unregister();
14     }
15 }
```

A 4.3 kódrészlet 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.



#### 4. FEJEZET: AZ OSGi SPECIFIKÁCIÓ

Abban az esetben ha több szolgáltatás implementálja ugyanazt az interfészt, szükség van egyedi azonosító használatára, melyet a 4.4 kódrészlet szemléltet.

Listing 4.4. Egyedi azonosító használata

```
1 public class Activator implements BundleActivator {
2
3     private ServiceRegistration serviceRegistration;
4
5     public void start(BundleContext context) throws Exception {
6
7         Hashtable<String, String> dictionary = new Hashtable<String, String>();
8
9         dictionary.put(Constants.SERVICE_DESCRIPTION, "This_is_a_Bundle");
10        dictionary.put("Name", "ExampleBundle");
11
12        serviceRegistration = context.registerService(Example.class.getName(), new
13            ExampleImpl(), dictionary);
14    }
15
16    public void stop(BundleContext context) throws Exception {
17
18        environmentServiceReg.unregister();
19    }
20 }
```

Megadható a `registerService()` módszernek egy `dictionary` paraméter amely, kulcs-érték párokat kell tartalmazzon. Így a szolgáltatás lekérésekor a 4.5 kódrészlet alapján hivatkozhatunk a szükséges szolgáltatásra.

Listing 4.5. Szolgáltatás lekérdezése egyedi azonosítóval

```
1 serviceReference = context.getServiceReferences(Example.class.getName(), "(Name=ExampleBundle)");
2 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. Ez azért probléma a kísérletek futtatása esetén, mert az egyik kísérlet beállít valamilyen értéket a környezetnek, mire a másik beállít egy teljesen mást. Így nem kaphatóak valós eredmények, mert amit az egyik tanuló algoritmus beállít az a másik elállítja. Fontos, hogy egy algoritmus végig tudja kísérni egy környezet állapotait, az általa megszabott szabályok szerint. A másik probléma, ha több különböző környezetről beszélünk, melyek teljesen más bemeneti értékekkel rendelkeznek. Ebben az esetben, ha a kevesebb bemeneti értékkel rendelkező környezet több bemeneti értéket kap, mint amennyire szüksége van, hibát fog eredményezni. Ennek a megoldására az OSGi keretrendszer definiál egy `ServiceFactory` interfészt, amely két metódussal rendelkezik, ahogyan a 4.6 kódrészlet is mutatja.

Listing 4.6. ServiceFactory

```
1 public class ExampleServiceFactory implements ServiceFactory {
2
3     public Object getService(Bundle bundle, ServiceRegistration registration) {
4
5         Example example = new ExampleImpl();
6
7         return example;
8     }
9 }
```

## 4. FEJEZET: AZ OSGi SPECIFIKÁCIÓ

```
10 public void ungetService(Bundle bundle, ServiceRegistration registration, Object service) {  
11  
12 }  
13  
14 }
```

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. Ezt szemlélteti a 4.7 kódrészlet.

Listing 4.7. Szolgáltatás regisztrálása `ServiceFactory` - val

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

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

### 4.3. A RoboRun projekt és az OSGi

A RoboRun projekt tervezésekor a hangsúly arra volt fektetve, hogy a készülő rendszer dinamikussága mellett, az egyes részek elkülönítődjenek egymástól. Másik fontos szempont az volt, hogy a készülő környezet teljesen egységes legyen, mely könnyen használható, bővíthető, módosítható és nem utolsósorban könnyen elérhető legyen. Ezen tulajdonságok meghatározása után, a rendszer megvalósításához a legjobb megoldásnak az OSGi használata tűnt, mely egy dinamikus, modularizált komponens modellt definiál komplex alkalmazások felépítésére. Az OSGi-t ötvözve a GlassFish alkalmazás szerver lehetőségeivel, minden adott volt a rendszer megvalósításához.

Ahhoz, hogy az OSGi konténerbe batyukat telepíthessünk, szükség van a projekt minden egyes batyuja esetén megadni a csomagolási típust, illetve különböző függőségeket, melyet a rendszer a konténerbe telepítés után használ. Amennyiben különböző függőségekkel rendelkezik egy batyu, a konténerbe telepítés pillanatában, az OSGi ellenőrzi, hogy kielégíthetőek-e ezek a függőségek. Ezen függőségeket, illetve a csomagolási típust is, a Maven által biztosított `pom.xml` állományban lehet megadni. A RoboRun projekt esetén minden szolgáltatás és batyu a `bundle` csomagolási formátumot kapta. Hiszen a szolgáltatások is `bundle` típusúak az OSGi-ban. Ezen szolgáltatások a `maven-archetype-quickstart` archetípusból lettek létrehozva. A Webes felületet biztosító csomag kiterjesztése `war` típusú, viszont ez is egy OSGi batyu, amely OSGi-ban a WAB (Web Application Bundle)-nak felel meg. A `RandomAgent` szolgáltatás esetén, a 4.8 kódrészlet szemlélteti a csomagolási típus megadásának módját.

Listing 4.8. Csomagolási típus

```
1 <groupId>edu.bbte</groupId>  
2 <artifactId>agent</artifactId>  
3 <version>0.0.1-SNAPSHOT</version>  
4 <packaging>bundle</packaging>
```

A `groupId` minden batyu esetén az `edu.bbte`, mely logikai csoportokba szervezi a komponenseket. Az `artifactId` minden batyut egyedi azonosítóval lát el. A projekt eleget tesz a Maven szabványának, miszerint egy projekten belül a `groupId` és `artifactId` párosítások egyediek kell legyenek. A

## 4. FEJEZET: AZ OSGi SPECIFIKÁCIÓ

version megadja a batyu aktuális verziójának számát és a packaging a csomagolási formátumot definiálja.

Maven esetén beszélhetünk direkt és tranzitív függőségekről. A direkt függőségek alatt értünk egy konkrét batyut vagy szolgáltatást melyet megadunk a pom.xml állományban. Tranzitív függőség alatt értjük azt, hogy egy direkt függőségként megadott függőség, függ más batyuktól vagy szolgáltatásoktól. A RoboRun projekt esetén, minden Agent komponens függ az Agent batyutól, mely egy interfészt definiál és minden Environment komponens függ az Environment batyutól. Ezen függőségekre a projekt fordításakor van szükség. Az RandomAgent függőségeit a 4.9 kódrészlet szemlélteti.

Listing 4.9. RandomAgent esetén a függőségek

```
1 <dependencies>
2   <dependency>
3     <groupId>org.osgi</groupId>
4     <artifactId>org.osgi.core</artifactId>
5     <version>4.3.0</version>
6     <scope>provided</scope>
7   </dependency>
8
9   <dependency>
10    <groupId>edu.bbte</groupId>
11    <artifactId>packages</artifactId>
12    <version>0.0.1-SNAPSHOT</version>
13  </dependency>
14
15  <dependency>
16    <groupId>edu.bbte</groupId>
17    <artifactId>agent</artifactId>
18    <version>0.0.1-SNAPSHOT</version>
19  </dependency>
20  <dependency>
21    <groupId>edu.bbte</groupId>
22    <artifactId>agentEnvironmentList</artifactId>
23    <version>0.0.1-SNAPSHOT</version>
24  </dependency>
25 </dependencies>
```

A futás idejű függőségek kielégítésére szükség van az OSGi által biztosított import-export mechanizmusra. Az import - export mechanizmus szintén a Maven által biztosított pom.xml - állományon keresztül került megvalósításra. Az Agent interfész esetén exportált illetve importált csomagok forráskódját 4.10 kódrészlet szemlélteti.

Listing 4.10. Az Agent interfész által importált és exportált csomagok

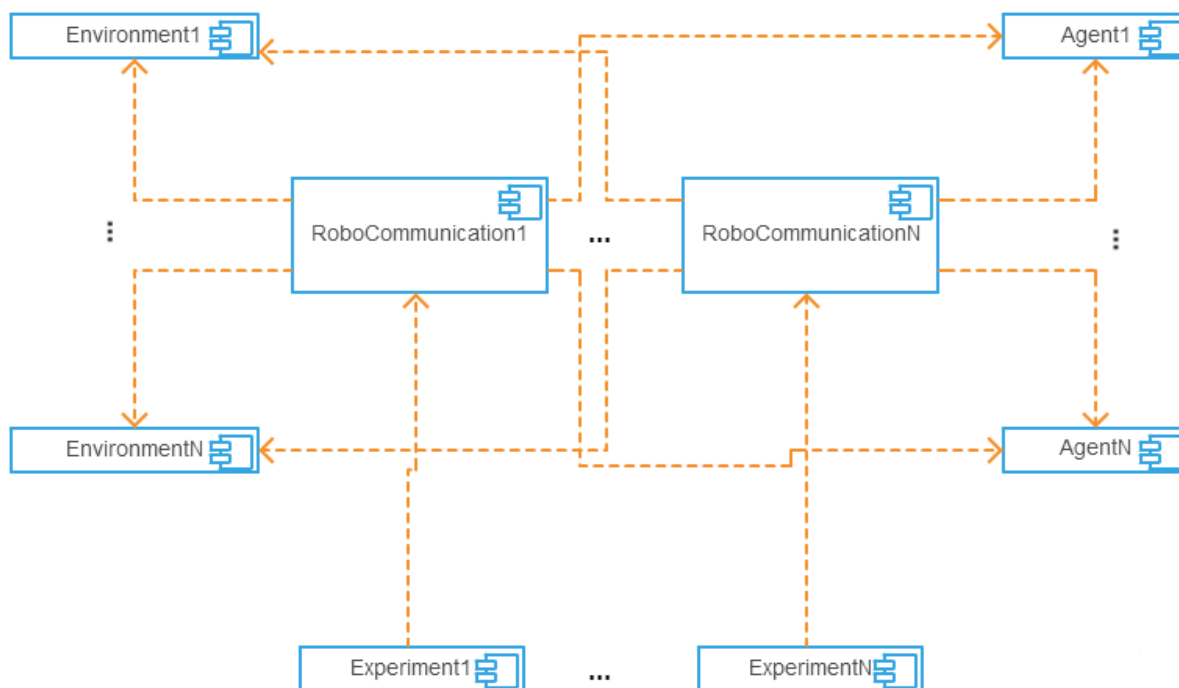
```
1 <Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
2 <Bundle-Version>${project.version}</Bundle-Version>
3 <Bundle-Activator>edu.bbte.agent.AgentActivator</Bundle-Activator>
4 <Export-Package>
5   edu.bbte.agent;version=${project.version}
6 </Export-Package>
7 <Import-Package>
8   edu.bbte.packages.types, *
9 </Import-Package>
```

Az Agent interfész által exportált csomag a edu.bbte.agent csomag és az általa importált csomag a edu.bbte.packages.types csomag. Az Agent interfésznek csak egy külső csomagra van szüksége a futáshoz. Viszont egyes szolgáltatásoknak több csomagra is szüksége lehet a futáshoz.

A RoboRun projekt esetében négy alap komponensről beszélhetünk. Ezen komponensek az Agent(Ügynök), az Environment(Környezet), az Experiment(Kísérlet), illetve a RoboCommunication(Kommunikációs réteg). E négy alapkomponekre épül a rendszer. Ezen komponensek az OSGi szabványnak megfelelően, külön szolgáltatások, kivéve az Experiment, mely nem

#### 4. FEJEZET: AZ OSGi SPECIFIKÁCIÓ

egy szolgáltatás, hanem egy közönséges batyu, amelynek az a szerepe, hogy lekérdezze a már telepített szolgáltatásokat és elindítson egy tesztet. Minden szolgáltatás külön batyuban található, ezt 4.4 ábra szemlélteti.



4.4. ábra. Osgi Alapkomponensek

A Robo, az Agent és az Environment interfészek külön batyuk, amelyek exportálják a csomagukat, így más batyuk számára is elérhetőek. A RoboCommunication osztály, amely valójában egy OSGi szolgáltatás, implementálja a Robo interfészt. Ehhez hasonló módon, minden Agent implementálni fogja az Agent interfészt és minden Environment implementálni fogja az Environment interfészt. Amennyiben a rendszerbe telepítésre kerül egy olyan Agent, amely nem implementálja az Agent interfészt, használhatatlan lesz, hiszen ahhoz, hogy egy szolgáltatást le tudjunk kérdezni a rendszerből, ismernünk kell az általa implementált interfészt. Az Experiment komponens, egy közönséges batyu, amely importálja a többi szolgáltatás által közzétett csomagokat és meghatározza, mely szolgáltatásokkal szeretne dolgozni. Ezen komponensek interakciója révén képes a rendszer megerősítéses tanulási algoritmusok futtatására és tesztelésére. Az interfészek és a komponensek közti kapcsolatot a 5.1 ábra szemlélteti, amely a 5.2 fejezetben található.

Az interakció belépési pontja az Experiment batyu. Ahhoz, hogy a batyu telepítésre kerülhessen az OSGi konténerbe, szükség van a függőségei kielégítésére. Az Experiment komponens sok függőséggel rendelkezik, hiszen neki szüksége van arra, hogy ismerje a telepített Agent és Environment példányokat. Az Experiment ismerheti az összes rendszerbe telepített Agent és Environment példányt, viszont arra is lehetőség van, hogy csak az éppen aktuális teszthez szükséges Agent és Environment példányok kerüljenek megadásra, mint függőség. Ez esetben, csak a függőségként meg-

#### 4. FEJEZET: AZ OSGi SPECIFIKÁCIÓ

adott Agent és Environment példányokkal képes dolgozni. Ezen függőségek mellett még szüksége van a RoboCommunication függőségre is, hiszen a RoboCommunication komponens által valósul meg a kommunikáció az egyes komponensek közt. Ezen függőségeken kívül, az Experiment komponensnek szüksége van ismerni, ezen szolgáltatások interfészeit is.

A szolgáltatások lekérése a szolgáltatás tárolóból a BundleContext referencián keresztül kerül megvalósításra. A batyuk implementálják a BundleActivator interfészt, mely a start(BundleContext context) és stop(BundleContext context) metódusokkal rendelkezik. Amikor telepítésre kerül egy batyu, akkor a start(BundleContext context) metódus automatikusan meghívásra kerül. Az egyes szolgáltatások lekérdezéséhez két lépésből áll. Első lépésben szükséges lekérdezni a szolgáltatás referenciát a getServiceReference metódus segítségével. Második lépésben szükséges lekérdezni a szolgáltatás objektumot a szolgáltatás referencia segítségével, mely a getService metódus által valósul meg. A RandomAgent szolgáltatás lekérdezését egy Experiment komponensben a 4.11 kódrészlet szemlélteti.

Listing 4.11. Szolgáltatás lekérdezése

```
1      agentServiceReference =  
2          context.getServiceReferences(Agent.class.getName(), "(AgentName=  
           RandomAgent)");  
3      agent = (Agent) context.getService(agentServiceReference  
           [0]);
```

A getServiceReferences metódus két paramétert kap. Az első paraméter az interfész neve, melyet a lekérdezni kívánt szolgáltatás implementál. A RandomAgent szolgáltatás esetén, az Agent interfész. A második paraméter egy egyedi azonosító, mely által megkülönböztethetők azon szolgáltatások, melyek ugyanazt az interfészt implementálják. A getService metódus, a kikért referencia értéket várja paraméterként. A getService által szolgáltatott objektum használata, teljesen megegyezik a standard Java objektumok használatával. Az objektumon keresztül meghívhatóak különböző metódusok, lekérdezhetőek és beállíthatóak értékek.

Az OSGi-ban az egyes szolgáltatások megőrzik az állapotukat egészen addig, amíg a rendszer fut vagy nem telepítik újra őket a konténerbe. A RoboRun projekt esetén, ez nem megfelelő, hiszen a projekt egy fontos alapkövetelménye, hogy párhuzamosan több teszt is legyen futtatható. Amennyiben egy szolgáltatás megőrzi az állapotát, ez azt jelentené a RoboRun projekt esetében, hogy a felhasználó elindít egy tesztet az általa választott Agent és Environment példánnyal, melyek a RoboCommunication szolgáltatáson keresztül valósítják meg a kommunikációt, és megvárja az eredményt, mely helyes eredmény fog a felhasználónak megadni. Majd indít még egy tesztet, de ebben az esetben, mindhárom komponens az előző teszt értékeivel fog rendelkezni, amely hibás teszt eredményekhez vezet. E probléma kiküszöbölése érdekében, került használatra a ServiceFactory interfész, amely az OSGi egy beépített interfésze. Így minden egyes alapszolgáltatás a RoboRun projekt esetén, implementálja a ServiceFactory interfészt, mely lehetővé teszi azt, hogy minden egyes szolgáltatás lekérdezéskor, a szolgáltatás egy teljesen új példányát téríti vissza a lekérdező batyu számára. A RandomAgent szolgáltatás esetén a ServiceFactory interfész implementációját a 4.12 szemlélteti.

Listing 4.12. RandomAgentServiceFactory osztály

```
1 public class RandomAgentServiceFactory implements ServiceFactory {
2
3     private static final Logger logger = Logger.getLogger(RandomAgentServiceFactory.class.
4         getSimpleName());
5     private int usageCounter = 0;
6
7     public Object getService(Bundle bundle, ServiceRegistration registration) {
8         usageCounter++;
9
10        logger.log(Level.INFO, "Create_object_of_RandomAgent_for_" + bundle.getSymbolicName());
11        logger.log(Level.INFO, "Number_of_bundles_using_service_" + usageCounter);
12
13        Agent agent = new RandomAgent();
14
15        return agent;
16    }
17
18    public void ungetService(Bundle bundle, ServiceRegistration registration, Object service) {
19
20        usageCounter--;
21
22        logger.log(Level.INFO, "Release_object_of_RandomAgent_for_" + bundle.getSymbolicName());
23        logger.log(Level.INFO, "Number_of_bundles_using_service_" + usageCounter);
24    }
25 }
```

A 13-adik sorban látható a `RandomAgent` osztály példányosításra, melyet a 15 -sorban a `getService()` osztály visszatérít. Ez által valósul meg az, hogy mindig egy új példány kerül visszatérítésre.

## 5. fejezet

# A rendszer felépítése és használata

***Összefoglaló:** E fejezet célja részletesen ismertetni a rendszer teljes architektúráját. A fejezet második részében a rendszer használatának ismertetése található.*

### 5.1. Alapelképzelés

A RoboRun projekt alapelképzelése egy olyan szoftver megalkotása, amely teljesen önállóan képes futtatni megerősítéssel tanulóval kapcsolatos algoritmusokat. Ezen algoritmusok futtatása, egy távoli szervergépen történik, amely folyamatosan elérhető. A tesztek futtatása szempontjából fontos a gyorsaság és a megbízhatóság. Ezen tulajdonságok mellett még szükség van arra, hogy a rendszer dinamikusságot kölcsönözzön az egyes komponenseknek. Minden nagyobb rész különálló komponensként legyen megvalósítva, a könnyebb átláthatóság, tesztelhetőség, módosíthatóság érdekében. Fontos tulajdonság, hogy a rendszer képes legyen párhuzamosan több teszt futtatására. A teszt eredmények megfelelően el kell legyenek tárolva egy központi adatbázisban. Az adatbázisban található bejegyzések hozzáférését egy webes felület biztosítja. A webes felület az adatbázis bejegyzések hozzáférése mellett, információkat szolgáltat a rendszerbe telepített aktív Agent és Environment példányokról illetve a tesztek állapotairól.

A webes felületen a felhasználónak lehetősége van megtekinteni a rendszerbe telepített és használható Agent és Environment példányokat, melyek alapján új Experiment-eket lehet implementálni. Megtekinthetők az aktív tesztek állapotai és különböző fontos információk az aktív tesztekéről, például: indítás ideje, aktuálisan hol tart, milyen Agent illetve Environment példánnyal fut a teszt. A webes felületen megtekinthetők a már befejeződött tesztek által generált adatok illetve statisztikai a hozzájuk tartozó statisztika.

A rendszer jelenlegi állapota tükrözi az alapelképzelések által megfogalmazott elvárásokat.

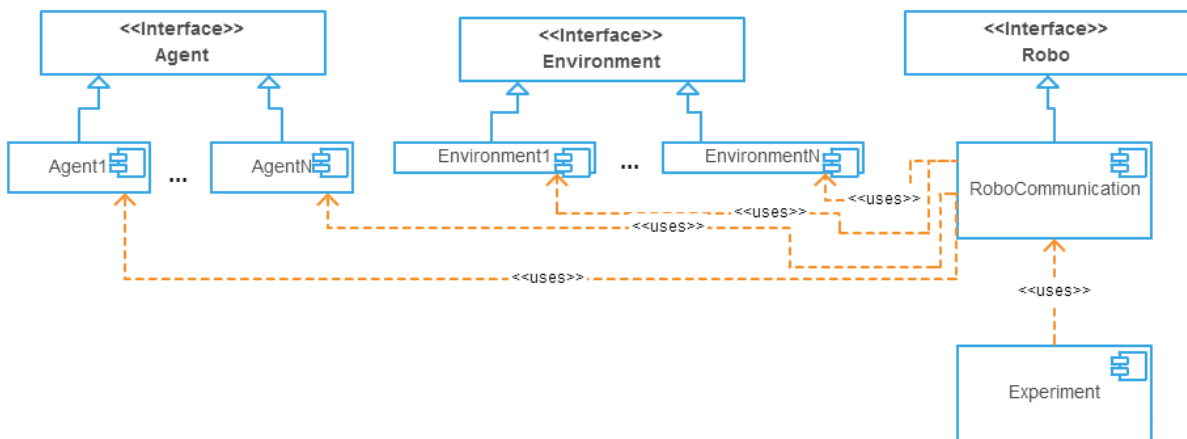
### 5.2. A rendszer felépítése

A RoboRun projekt négy alkomponensre épül. Melyek az Agent, Environment, Experiment és RoboCommunication komponensek formájában vannak megvalósítva. Az Agent komponens valósítja meg a mesterséges tanulási algoritmusokat. Az Environment a környezet, amelyet az Agent próbál megtanulni helyesen használni. Az Experiment az egyes tesztek belépési pontjaként szolgál, ő határozza meg az epizódok és az epizódonkénti maximális lépések számát. A RoboCommunication

## 5. FEJEZET: A RENDSZER FELÉPÍTÉSE ÉS HASZNÁLATA

komponens felelős az előbbi három komponens között teljes kommunikáció lebonyolításáért egy teszt során, melyet a 2.2 ábra szemléltet.

Az Experiment komponens kivételével, minden alkomponens egy neki megfelelő interfészt implementál. Ez által kerül megvalósításra az, hogy a rendszerben egyszerre több Agent és Environment példány lehet telepítve, illetve ezek cseréje és módosítása is lényegesen könnyebb. A RoboCommunication komponensből egy található a rendszerben, viszont ez is könnyedén módosítható vagy cserélhető anélkül, hogy az egyéb komponenseket módosítani kellene. A RoboRun projekt esetén az alkomponensek és a hozzájuk tartozó interfészeket a 5.1 ábra szemlélteti.

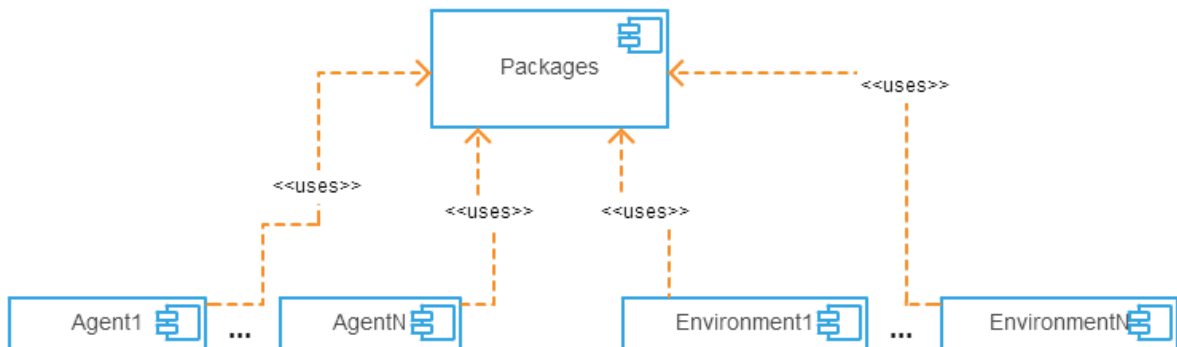


5.1. ábra. A RoboRun projekt alkomponensei és interfészek

Az alkomponenseknek szükségük van néhány saját típusra a tesztek futtatásához. Ezen típusok egy külön batyuban kaptak helyet, melynek neve packages. Ezen típusok az R1-Glue projektből lettek átvéve és felhasználva módosítás nélkül. A packages batyu közléteszi a különböző csomagjait, amelyet az alkomponensek importálnak. A packages batyu által közzétett csomagok:

taskSpec - amely meghatározza a tesztek során az egyes lépésekhez szükséges adatokat.

types - amely meghatározza a tesztekhez tartozó típusokat, például az Action vagy az Observation típusok.



5.2. ábra. Az alkomponensek által felhasznált komponens



## 5. FEJEZET: A RENDSZER FELÉPÍTÉSE ÉS HASZNÁLATA

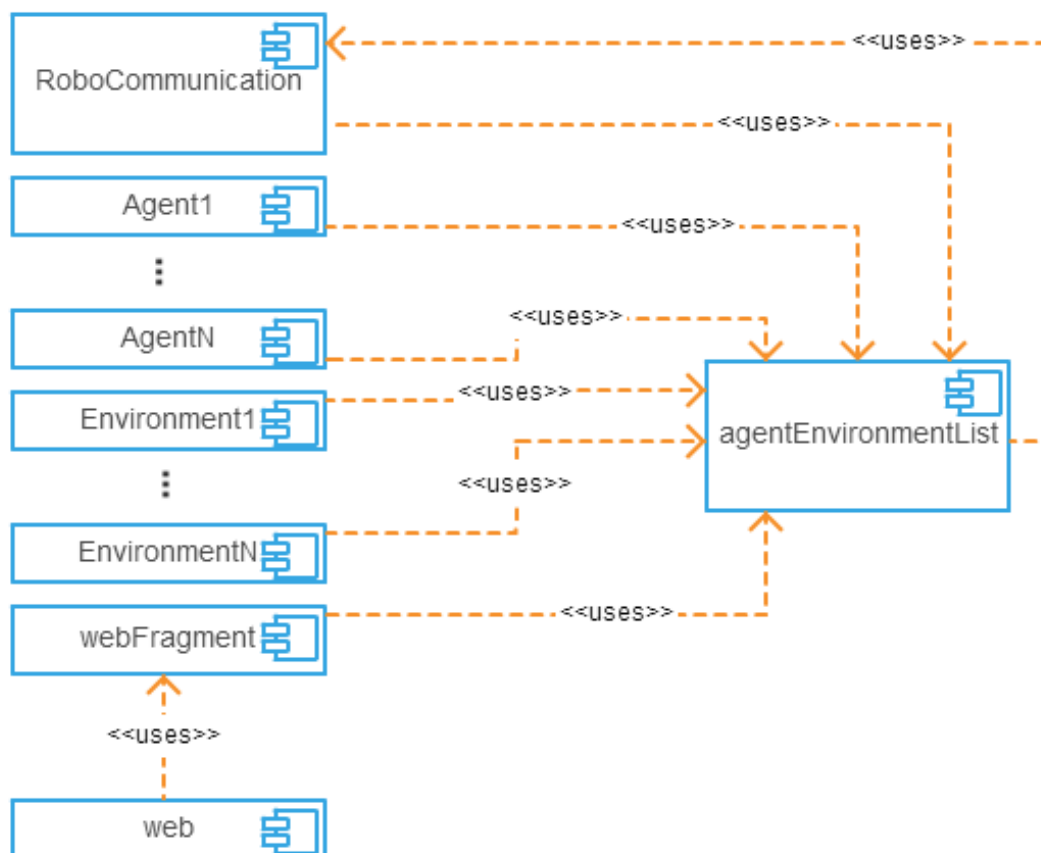
Az adat hozzáférési réteg szintén egy batyuként van definiálva, amely egy különálló komponensként működik. Legfőbb előnye szint a könnyen cserélhetőség. Amennyiben bármilyen más implementációra van szükség a rendszer és az adatbázis között, e réteg könnyen cserélhető akár a teljes rendszer leállítása nélkül is. Az adat hozzáférési réteg a `dataModel` nevet kapta, mely közzéteszi az `edu.bbte.dataModel` csomagját. Ezt a csomagot a `RoboCommunication` komponens fogja importálni és használni. A `RoboCommunication` komponens rendelkezik a egy teszt futtatása során az összes olyan információval, amelyre szükség van egy adatbázis bejegyzéshez. Az adatbázisba bekerülő adatok egy teszthez a következők: az `Agent` neve, az `Environment` neve, a teszt elkezdésének ideje, illetve két fájl. A fájlok nevei a teszt indításának ideje(ÉvHónapNapÓraPerc + egy véletlenszerűen generált szám + Results vagy Stat) formátumúak. A Results végződésű fájl tartalmazza a teszt alatt létrejött összes adatot, a Stat végződésű fájl statisztikai adatokat tartalmaz a tesztekéről. A fájlok a webes felület segítségével érhetőek el, illetve lehetőség van a fájlok letöltésére. A 5.3 ábra szemlélteti a kapcsolatot az adat hozzáférés réteg és a `RoboCommunication` komponensek között.



5.3. ábra. Az adathozzáférési réteg és a `RoboCommunication`

A webes megjelenítéshez szükség van egy olyan komponensre, amely összegyűjti az adatokat a többi komponensről és ezt átadja a webes felületnek. Erre a feladatra a `agentEnvironmentList` komponens van kijelölve. E batyu folyamatosan értesül arról, hogyha új `Agent` vagy `Environment` példánnyal gazdagodik a rendszer, illetve arról is ha törlésre kerül valamelyik. Az `agentEnvironmentList` komponens közzéteszi az `edu.bbte.agentEnvironmentList` csomagját, melyet minden `Agent` és `Environment` komponens importál, így beleírhatják magukat induláskor és törölhetik magukat leálláskor. Ezek megvalósítása az `Activator` osztályok `start()` és `stop()` metódusaiban történik, annak érdekében, hogy biztosan végrehajtsódjon a listába való írás illetve törlés. E komponens feladatai közé tartozik az is, hogy a webes felületet tájékoztassa az éppen aktuálisan futó tesztek állapotairól. Ehhez szükséges az, hogy `RoboCommunication` komponens is importálja az `edu.bbte.agentEnvironmentList` csomagot, mely rendelkezik egy `AgentEnvironment` osztállyal. Az `AgentEnvironment` osztály `addTest()` metódusának utolsó paramétere egy `Robo` típusú objektum, mely az aktuális teszt `RoboCommunication` osztály egy példánya. Így e példány `getPercent()` metódusa által lekérhető az aktuális teszt állapota. Ahhoz, hogy a webes felülethez eljusson az információ, szükség van egy köztes rétegre, amely a `webFragment` nevet viseli. A `webFragment` komponens szintén egy batyu az `OSGi` konténerben, amely elkéri a szükséges információkat az `agentEnvironmentList` komponensről és ezt átadja a `web` komponensnek megjelenítésre. Ezen komponensek kapcsolatáról a 5.4 ábra nyújt összefogóbb

képet.



5.4. ábra. A webes felület megjelenítéséért felelős komponensek

### 5.2.1. Webes felület

A RoboRun projekt webes felületének megvalósítása a Vaadin keretrendszer által biztosított komponensek felhasználásával történik. A megjelenítésért a web komponens felelős, mely rendelkezik egy `FragmentFactory` interfésszel. A `FragmentFactory` interfésznek egyetlen egy metódusa van, a `getFragment()` metódus, amely egy `com.vaadin.ui.Component` típusú komponenszt térít vissza. Ezt az interfészt implementálja a `webFragment` komponens. Erre az interfészre azért van szükség, mert így megoldható, hogy a webes felület egyszerre több fragmentből álljon össze. A RoboRun projekt esetén csak egy fragmentre van szükség, viszont bármilyen új funkció úgy hozzáadható a webes felülethez, hogy a régieket kellene módosítani, illetve anélkül, hogy a rendszert újra kellene indítani. Ezzel is megkönnyítve a továbbfejlesztést. Ahhoz, hogy a fragmentek könnyedén telepíthetők legyenek, szükség van használni az OSGi által kínált `ServiceTracker`<sup>1</sup> osztályt, mely a `ServiceTrackerCustomizer` interfészt implementálja. Három metódussal rendelkezik az interfész. Az `addinService()` metódussal mely egy szolgáltatás referenciát vár paraméterként, il-

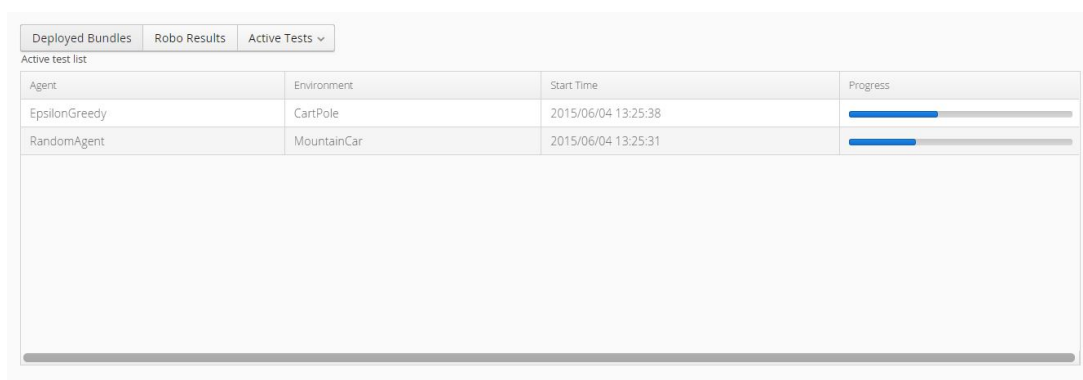
1. <https://osgi.org/download/r6/osgi.core-6.0.0.pdf>

## 5. FEJEZET: A RENDSZER FELÉPÍTÉSE ÉS HASZNÁLATA

letve a `modifiedService` és a `removedService`, amelyek egy szolgáltatás referenciát és egy szolgáltatást várnak paraméterként. A `ServiceTracker` - osztály példányosítása a `web` komponens `VaadinActivator` osztályában történik. A példányosítás után közvetlen megnyitásra kerül a `ServiceTracker`. A megnyitást követően, a webes felület értesül arról, ha új fragment kerül telepítésre, illetve arról, ha egy fragmentet töröltek, így az oldal újra betöltése után már használhatóak is az új funkciók.

### 5.2.2. Webes felület funkciói

Az oldal betöltődését követően a felhasználó az aktív tesztek állapotairól szerezhet tudomást. Amennyiben nincsenek aktív tesztek a felhasználó a "There is no active test available!" felirattal találkozik. Amennyiben vannak aktív tesztek a 5.5 ábra szemlélteti a nézetet. Az aktív tesztekéről látható, hogy melyik `Agent` és `Environment` példánnyal dolgozik az adott teszt. Látható a teszt indításának ideje, illetve az, hogy a teszt éppen hol tart. Ezt a "Progress" oszlop szemlélteti.



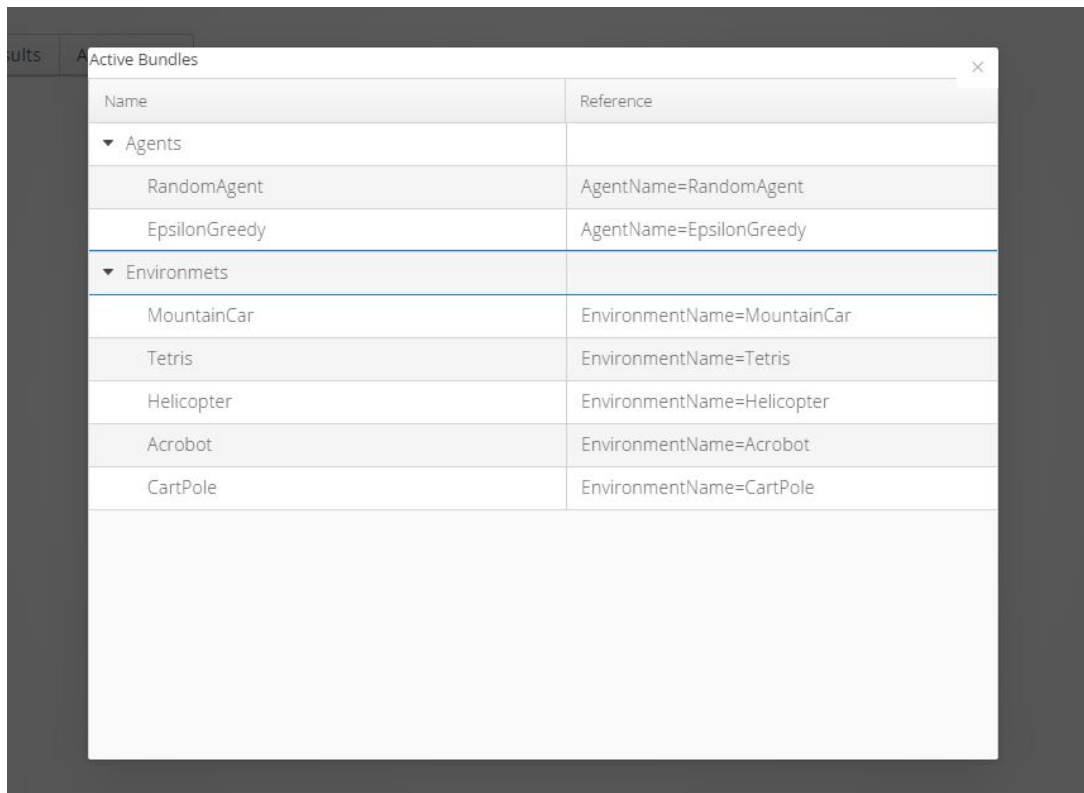
Agent	Environment	Start Time	Progress
EpsilonGreedy	CartPole	2015/06/04 13:25:38	<div><div></div></div>
RandomAgent	MountainCar	2015/06/04 13:25:31	<div><div></div></div>

5.5. ábra. Aktív tesztek nézet

Az "Deployed Bundles" menüpont alatt találhatóak a rendszerbe telepített `Agent` és `Environment` példányok listája. A nézet két oszlopot tartalmaz. Az első oszlopban találhatóak a telepített `Agent` és `Environment` példányok nevei, míg a második oszlopban található, hogy hogyan hivatkozhatunk rájuk, amennyiben szeretnénk lekérdezni valamelyik szolgáltatást. Az `Agent` és `Environment` példányok csoportosítva vannak. Az "Agent" fül és az "Environment" is lenyitható. Ezt a 5.6 kép szemlélteti.

Az tesztek összesítő adatait illetve a statisztikai adatokat a "Robo Results" menüpont alatt érhetjük el. Itt található egy táblázat, melynek egy sora megfelel egy adatbázis bejegyzésnek. Az első oszlop az `Agent` nevét tartalmazza, a második oszlop az `Environment` nevét tartalmazza, a harmadik oszlop a `Start time`, mely megadja a teszt indításának idejét. A negyedik oszlop az `All Result`, amely az aktuális teszt által generált összes információt tartalmazza. Az ötödik oszlop a `Statistics`, amely az aktuális teszthez tartozó statisztikai adatokat tartalmazza. Az adatok megtekintéséhez szükség van letöltésre. A letöltést "Click to download" gombra kattintva lehet megkezdeni. E nézetet a 5.7 kép szemlélteti.

## 5. FEJEZET: A RENDSZER FELÉPÍTÉSE ÉS HASZNÁLATA



The screenshot shows a window titled 'Active Bundles' with a close button in the top right corner. The window contains a table with two columns: 'Name' and 'Reference'. The table is divided into two sections: 'Agents' and 'Environments'. The 'Agents' section lists 'RandomAgent' and 'EpsilonGreedy'. The 'Environments' section lists 'MountainCar', 'Tetris', 'Helicopter', 'Acrobot', and 'CartPole'.

Name	Reference
▼ Agents	
RandomAgent	AgentName=RandomAgent
EpsilonGreedy	AgentName=EpsilonGreedy
▼ Environments	
MountainCar	EnvironmentName=MountainCar
Tetris	EnvironmentName=Tetris
Helicopter	EnvironmentName=Helicopter
Acrobot	EnvironmentName=Acrobot
CartPole	EnvironmentName=CartPole

5.6. ábra. Aktív Agent és Environmentek nézet

### 5.3. A rendszer telepítése

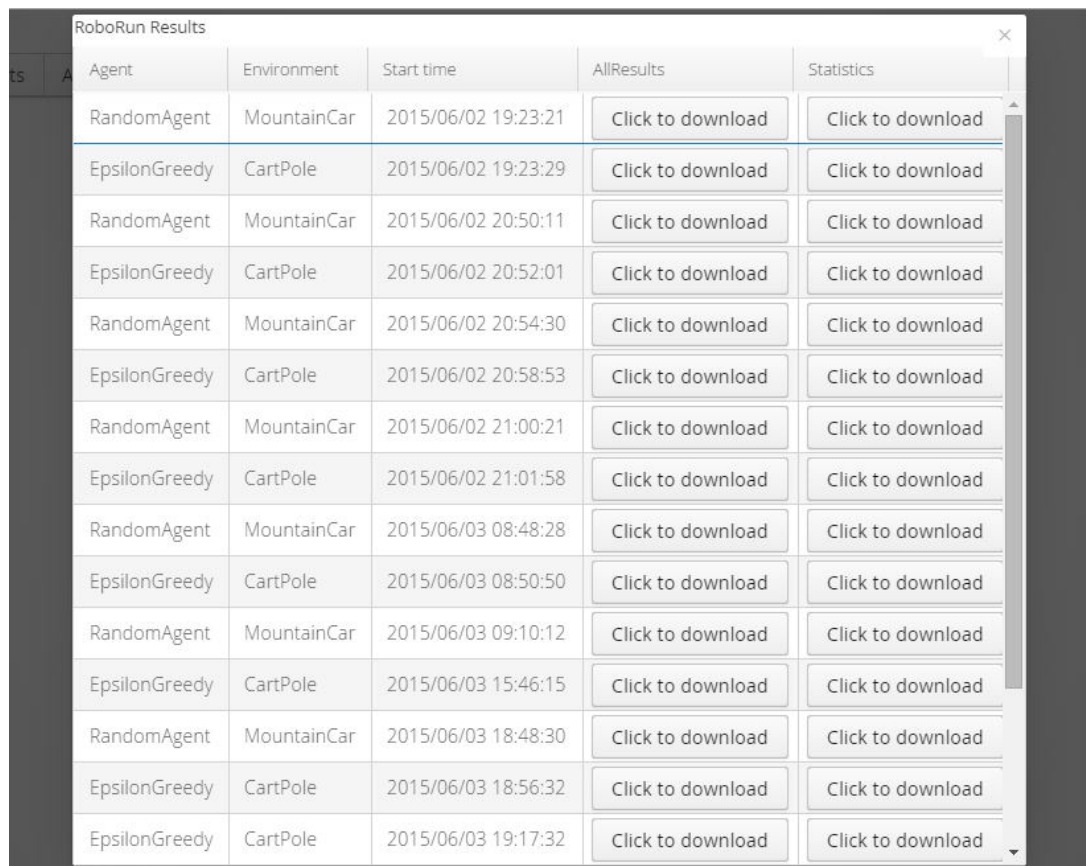
A rendszer telepítéséhez szükség van egy GlassFish alkalmazás szerverre és egy adatbázis kezelő rendszerre. A rendszer telepítése egyszerű, hiszen a komponensek különálló egységek. Az egyetlen dolog amire figyelni kell, a komponensek telepítésének sorrendje, mivel az egyes komponensek függnek más komponensektől.

A komponensek telepítésének sorrendje:

- packages - amely tartalmazza a szükséges típusokat
- agent - amely az Agent példányok interfészét tartalmazza
- environment - amely az Environment példányok interfészét tartalmazza
- robo - amely a RoboCommunication komponens interfészét tartalmazza
- dataModel - az adatbázis hozzáférési réteget tartalmazza
- agentEnvironmentList - amely a webes felülethez szükséges adatokat gyűjti össze a komponensektől
- web - amely a webes felület megjelenítéséért felelős
- webFragment - amely a webes felülethez szolgáltat funkcionálisokat
- RoboCommunication - amely az egyes Agent és Environment példányok közötti kommunikációért felelős

## 5. FEJEZET: A RENDSZER FELÉPÍTÉSE ÉS HASZNÁLATA

Ezen komponensek telepítése után következhet az Agent és Environment példányok telepítése, majd az Experiment példányok telepítése, amelyek elindítanak egy tesztet.



The screenshot shows a window titled "RoboRun Results" with a table of test results. The table has five columns: Agent, Environment, Start time, AllResults, and Statistics. The data is organized into rows, alternating between MountainCar and CartPole environments, each tested with RandomAgent and EpsilonGreedy agents. Each row includes a timestamp and a "Click to download" button for both the results and statistics.

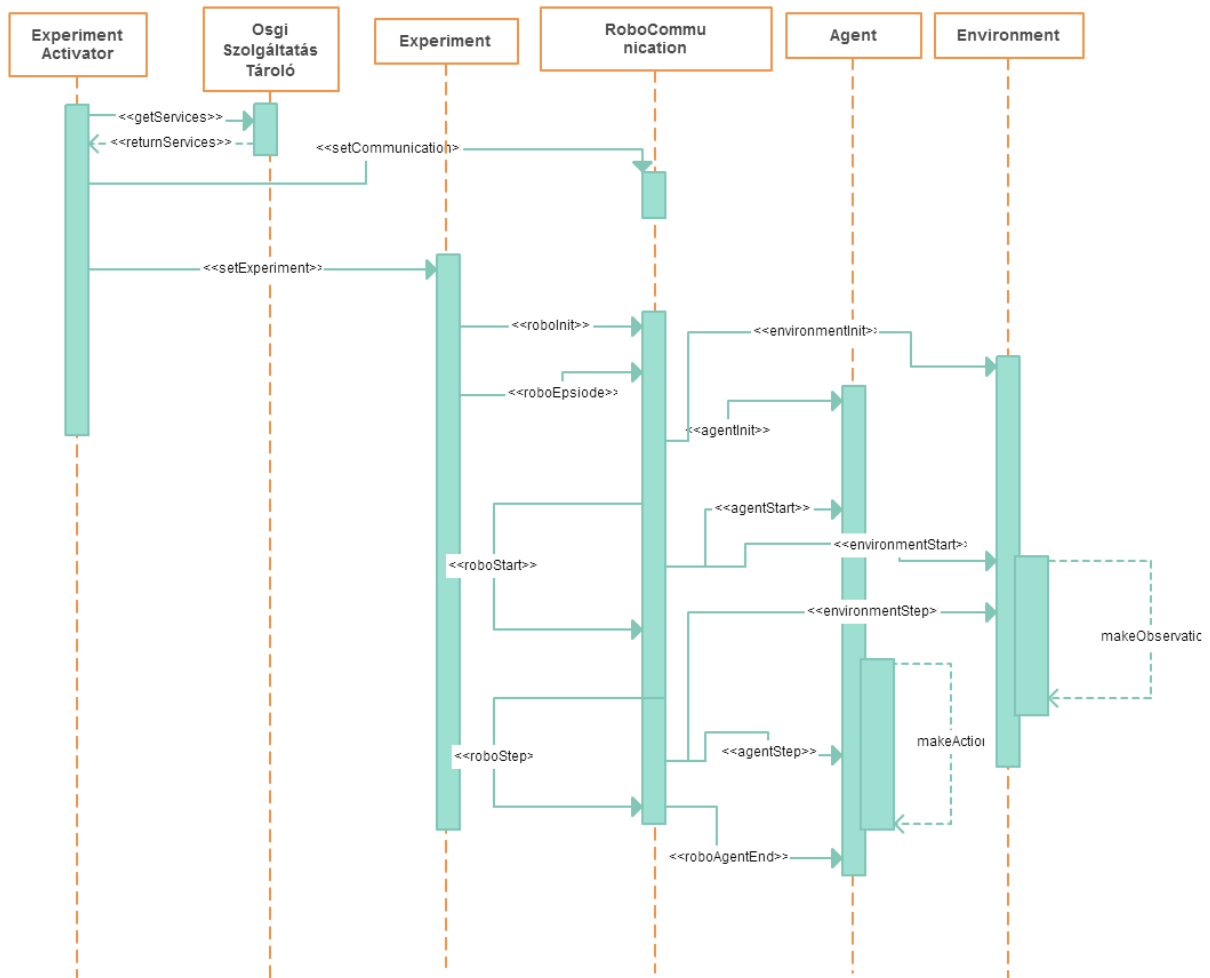
Agent	Environment	Start time	AllResults	Statistics
RandomAgent	MountainCar	2015/06/02 19:23:21	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/02 19:23:29	Click to download	Click to download
RandomAgent	MountainCar	2015/06/02 20:50:11	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/02 20:52:01	Click to download	Click to download
RandomAgent	MountainCar	2015/06/02 20:54:30	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/02 20:58:53	Click to download	Click to download
RandomAgent	MountainCar	2015/06/02 21:00:21	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/02 21:01:58	Click to download	Click to download
RandomAgent	MountainCar	2015/06/03 08:48:28	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/03 08:50:50	Click to download	Click to download
RandomAgent	MountainCar	2015/06/03 09:10:12	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/03 15:46:15	Click to download	Click to download
RandomAgent	MountainCar	2015/06/03 18:48:30	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/03 18:56:32	Click to download	Click to download
EpsilonGreedy	CartPole	2015/06/03 19:17:32	Click to download	Click to download

5.7. ábra. A teszt eredményeket tartalmazó nézet

### 5.4. Tesztek futtatásának folyamata

Minden teszt futtatása az Experiment komponenstől indul. Az Experiment komponens Activator osztálya lekérdezi a szükséges Agent, Environment és RoboCommunication példányokat, majd a RoboCommunication szolgáltatás setCommunication() metódusának átadja a lekérdezett Agent és Environment példányokat, illetve ezek neveit String-ként. Majd meghívja az Experiment osztály setExperiment metódusát, melynek átadja a RoboCommunication szolgáltatást. Az Experiment osztályban először meghívásra kerül a RoboCommunication szolgáltatás példány roboInit() metódusa, mely beállítja a szükséges bemeneti értékeket az Environment számára, illetve átadja a bemenetek listáját az Agent - nek, mely meghatározza az első lépésben végrehajtandó műveletek halmazát. Ezt követően az Experiment meghatározza az epizódok és a lépések számát, majd ezekkel az értékekkel meghívásra kerül a RoboCommunication szolgáltatás roboEpisode() metódusa. A roboEpisode() metódus meghívja a roboStart() metódust, mely elindítja az Environment környezetet, azaz kezdeti pozícióba helyezi. Az Environment a

kezdeti pozícióról alkot egy megfigyelést, melyet visszatérít. A megfigyelés alapján a tanuló algoritmus is elindulhat. Tehát az Agent a megfigyelés alapján generál egy művelet halmazt. Amennyiben az első lépés nem vezetett a végső állapothoz, megkezdődik a léptetés. Amely abban áll, hogy folyamatosan a tanuló algoritmus generál egy művelet halmazt melyet, a környezet végrehajt és megfigyeléseket ad vissza a tanuló algoritmus számára. A tanuló algoritmus a megfigyelések különböző jutalmakkal lesznek ellátva. Ezek alapján a tanuló algoritmus próbál minél optimálisabb művelet halmazt generálni a környezet számára. Egy teszt futása, addig tart amíg nem ér az epizódok végére. Viszont epizódonként addig tart egy teszt amíg nem éri el a lépések maximális számát, vagy amíg terminális állapotba nem kerül. A tesztek futtatása során az a cél, hogy minél több epizódban, minél kevesebb lépés végrehajtása után terminális állapotba kerüljön a környezet. Ezt szemlélteti a 5.8 ábra.



5.8. ábra. Egy teszt életciklusa

## 6. fejezet

# Következtetés és továbbfejlesztési lehetőségek

A RoboRun projekt keretein belül megvalósításra került egy megerősítéses tanulási algoritmusok tesztelésére szolgáló teljes rendszer amely lehetőséget biztosít a folyamatos és könnyed használatra. A rendszer dinamikus, megvalósítja a komponensek egymástól való elválasztását. A projekthez tartozik egy webes felület melyen megtekinthetők a telepített batyuk, az aktív tesztek, illetve a már lefuttatott tesztek eredményei. A rendszer ezen eredményeket egy adatbázisban tárolja. A RoboRun projekt teljesen az OSGi alkalmazás modellre épül.

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

Az OSGi alkalmazás modell által minden komponens külön van választva, így a már meglévő részek rugalmasan továbbfejleszthetők és kiegészíthetők. A RoboRun projekt jelen formájában egy OSGi konténerben van telepítve, mely egy GlassFish szerveren fut.

A projekt továbbfejlesztésére számos lehetőség létezik. Egyik legfontosabb ilyen lehetőség, a projekthez egy Eclipse Plugin[4] 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` komponens a saját gépről futtatható legyen, míg a rendszer többi része egy központi elérésű szerveren található. Ehhez szükség van felhasználni az OSGi által biztosított OSGi Remote Services tulajdonságot. Ez azt jelenti, hogy az OSGi képes távoli metódushívásokra, oly módon, hogy a rendszer egyik része fut egy OSGi konténerben, míg a rendszer többi része egy teljesen más OSGi konténerben található. Az Eclipse Plugin esetén az Eclipse fejlesztői környezet lenne az egyik OSGi konténer, míg a másik a GlassFish szerverre telepített lenne. Így az erőforrás igényes komponensek, mint például a `Agent`, `Environment` és a `RoboCommunication` komponensek a GlassFish szerveren futnának és az `Experiment` komponens futna az Eclipse fejlesztői környezetből, amely egy saját lokális számítógépen található. Egy másik továbbfejlesztési lehetőségként érdemes megemlíteni az RMI(Remote Method Invocation) kommunikációt. Mely kapcsolatot létesít a szerverrel és képes lekérdezni, különböző objektumokat. Ezzel az a probléma, hogy a lekérdezett objektumok szerializálása kerülnek elküldés előtt és az OSGi szolgáltatásokat nem lehet szerializálni a konténeren kívülre, mert ezen szolgáltatások csak a konténeren belül elérhetőek. Amennyiben erre a problémára sikerülne találni megoldást, ez a továbbfejlesztés nagyon egyszerű és hasznos módja lenne, hiszen ez által szintén teljesen

## 6. FEJEZET: KÖVETKEZTETÉS ÉS TOVÁBBFEJLESZTÉSI LEHETŐSÉGEK

áthelyezhető az `Experiment` komponens saját lokális számítógépre.

A rendszer jelenleg rendelkezik néhány alap környezettel és két tanuló algoritmussal. Viszont ezek implementálása nem a projekt része. Fontos továbbfejlesztési lehetőség lehet, újabb környezetek és tanulási algoritmusokkal kiegészíteni a jelenlegi rendszert, melyek hozzáadása a rendszer tervezésének és felépítésének köszönhetően egyszerűen eszközölhető.

A rendszer továbbfejlesztését a webes felülettel lehetne folytatni, amely jelen formájában egy prototípus és rendelkezik a legfontosabb alapfunkcionalitásokkal, melyek szükségesek ahhoz, hogy információkat kapjunk a rendszerről és a rendszer által futtatott tesztek állapotairól, illetve az ezek által generált adatok halmazáról. Ezen funkcionálisok könnyedén kibővíthetők a 5.2.1 alfejezetben leírt felépítése által.



# Irodalomjegyzék

- [1] Apache felix hivatalos weboldal. <http://felix.apache.org/>. Utolsó megtekintés dátuma: 2015-06-02.
- [2] Paller gábor - osgi és batyuk. <http://pallergabor.uw.hu/hu/java-app/OSGi.html>. Utolsó megtekintés dátuma: 2015-05-15.
- [3] *Dynamic Programming and Optimal Control*. Athena Scientific, 2005.
- [4] Eclipse plugin hivatalos weboldal. <http://marketplace.eclipse.org/>. Utolsó megtekintés dátuma: 2015-05-29.
- [5] Eclipse equinox hivatalos weboldal. <http://eclipse.org/equinox/>. Utolsó megtekintés dátuma: 2015-05-28.
- [6] Git hivatalos weboldal. <https://github.com/>. Utolsó megtekintés dátuma: 2015-06-08.
- [7] Glassfish hivatalos weboldal. <https://glassfish.java.net/>. Utolsó megtekintés dátuma: 2015-05-15.
- [8] Maven hivatalos weboldal. <https://maven.apache.org/>. Utolsó megtekintés dátuma: 2015-06-06.
- [9] Osgi hivatalos weboldal. <https://osgi.org/download/r6/osgi.core-6.0.0.pdf>. Utolsó megtekintés dátuma: 2015-05-15.
- [10] Projektmenedzsment. <http://hu.wikipedia.org/wiki/Projektmenedzsment>. Utolsó megtekintés dátuma: 2015-06-04.
- [11] Redmine hivatalos weboldal. <http://www.redmine.org/>. Utolsó megtekintés dátuma:

## IRODALOMJEGYZÉK

2015-06-07.

[12] *Reinforcement Learning: An Introduction*. A Bradford Book, The MIT Press, 2005.

[13] RL- glue hivatalos weboldal. [http://glue.rl-community.org/wiki/Main\\_Page](http://glue.rl-community.org/wiki/Main_Page).  
Utolsó megtekintés dátuma: 2015-06-05.

[14] Szolgáltatás orientált architektúra wikipedia. [http://hu.wikipedia.org/wiki/Szolgáltatásorientált\\_architektúra](http://hu.wikipedia.org/wiki/Szolgáltatásorientált_architektúra). Utolsó megtekintés dátuma: 2015-05-10.

[15] Tortoise git. <https://code.google.com/p/tortoisegit/>. Utolsó megtekintés dátuma: 2015-05-18.

[16] Vaadin hivatalos weboldal. <https://vaadin.com/home>. Utolsó megtekintés dátuma: 2015-06-06.