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. 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.

A FEJEZETES RÉSZ, MÉG AZ ALAPELGONDOLÁST ÍRJA LE. EZT MAJD AKKOR ÍROM ÁT HA KÉSZ AZ EGÉSZ!

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

A megerősítéses tanulás nem új keletű. Már az 1950 - es évek előtt foglalkoztak mesterséges intelligencia kutatással, csak akkor még nem így nevezték. Az 1950 - es években John McCarthy[1] megalkotja a mesterséges intelligencia kifejezést. Az emberek fantáziájában már nagyon rég benne van az, hogy vajon mi lenne ha számítógépek intelligensek lennének, vagy tudnának önállóan gondolkodni. Rengeteg kutatás és kísérletezés folyik ennek érdekében. A megerősítéses tanulásról megoszlanak a vélemények, hiszen egyes emberek szerint nem lenne jó ha a számítógépek önállóan tudnának gondolkodni. Mások szerint nagyon jó lenne, hiszen megkönnyítenék az emberek életét.

A gépi tanulás is a természetből indul ki. Az élő szervezetet próbálja modellezni. Ezen algoritmusok legfőbb jellemzője az adaptációs[2] tanulási képesség. A tanulás és az adaptáció valójában az élő szervezet működését jellemzik. Hiszen, ahogy 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[3] esetén minták alapján történik a tanulás. Ez azt jelenti, hogy nagy mennyiségű adatból próbálunk megfelelő mennyiségű ismeretet szerezni és ez által 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.

Kijelenthetjük, hogy ezen algoritmusok legfőbb jelmezője az, hogy nem előre meghatározott képességekkel rendelkeznek, amelyek csak egy adott feladat elvégzésére lenne elegendő, hanem képesek arra, hogy folyamatosan fejlesszék képességeiket és ez által képesek legyenek alkalmazkodni új és ismeretlen környezetekhez.

1. http://en.wikipedia.org/wiki/John_McCarthy_%28computer_scientist%29
2. http://hu.wikipedia.org/wiki/Adaptáció
3. http://hu.wikipedia.org/wiki/Neurális_hálózat

2.2. A megerősítéses tanulás algoritmusok kipróbálásának alap lépései

A megerősítéses tanulási algoritmusok állapotmegfigyeléseken és jutalmakon alapulnak. Ez szintén az élő szervezetekre vezethető vissza, hiszen az állatok tapasztalatainak megszerzése is az idegrendszer állapotmegfigyelésein alapszik. Megfigyelhető például a kutyák esetében, ha egy forró tárgyhoz hozza ér hamar elkapja a mancsát, viszont még néhányszor próbálkozik. Majd néhány próbálkozás után megtanulja, hogy egy adott tárgy ha forró ahhoz többet ne érjen hozza. Minden egyes algoritmus bizonyos számú epizódot hajt végre, mely rendelkezik bizonyos számú lépés sorozattal. A tesztek futtatásának a célja, hogy egy optimális stratégiát alakítson ki a feladat megoldására illetve, hogy maximalizálja a jutalmakat.

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 lépes 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 becsélés 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 epizódosok számát, illetve az epizodikusokon belül a lépések számát. E három fő részről és az ezek közötti kommunikáció részletesebb leírását az Alapfogalmak bevezetése Szekció tartalmazza.

2.2.1. Kihívások a kísérletek futtatása során

A megerősítéses tanulási algoritmusok futtatása saját számítógépen nem a legjobb megoldás. Hiszen ezen kísérletek futtatása egy időigényes folyamat, hiszen nagyon sok lépésre lehet szükség az egyes feladatok sikeres megoldásához. Tehát ahhoz, hogy sikeresen kialakításra kerüljön az optimális stratégia a feladat megoldására. A kísérletek futtatása során felmerül az a probléma, hogy egy adott kísérlet több példányát kellene végrehajtani egyidejűleg, illetve több különböző kísérlet példányait egyszerre. Hiszen, 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 tanulási 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 managelhetőek lennének a tesztek.

Ennek megoldására egy olyan szimulációs környezet megvalósítása kínál lehetőséget, amely egy távoli, jól felszerelt szervergépen fut, mely folyamatosan elérhető 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.

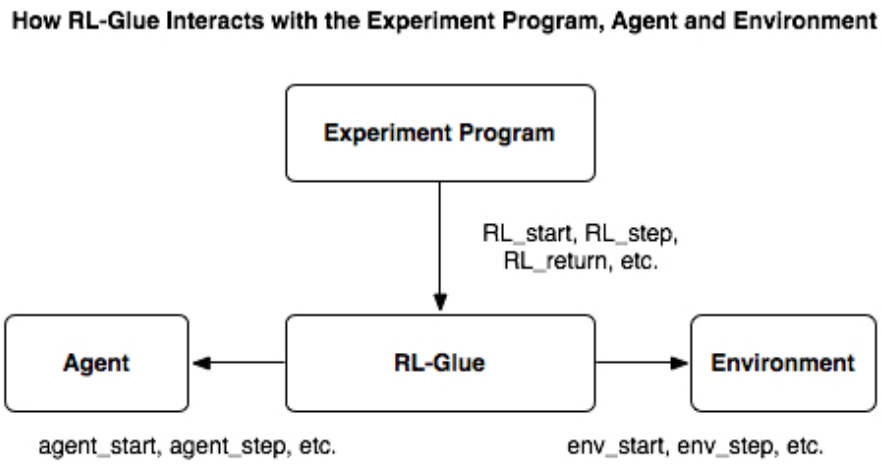
2.3. Alapfogalmak bevezetése

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 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 módon 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.

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 úgyneve-



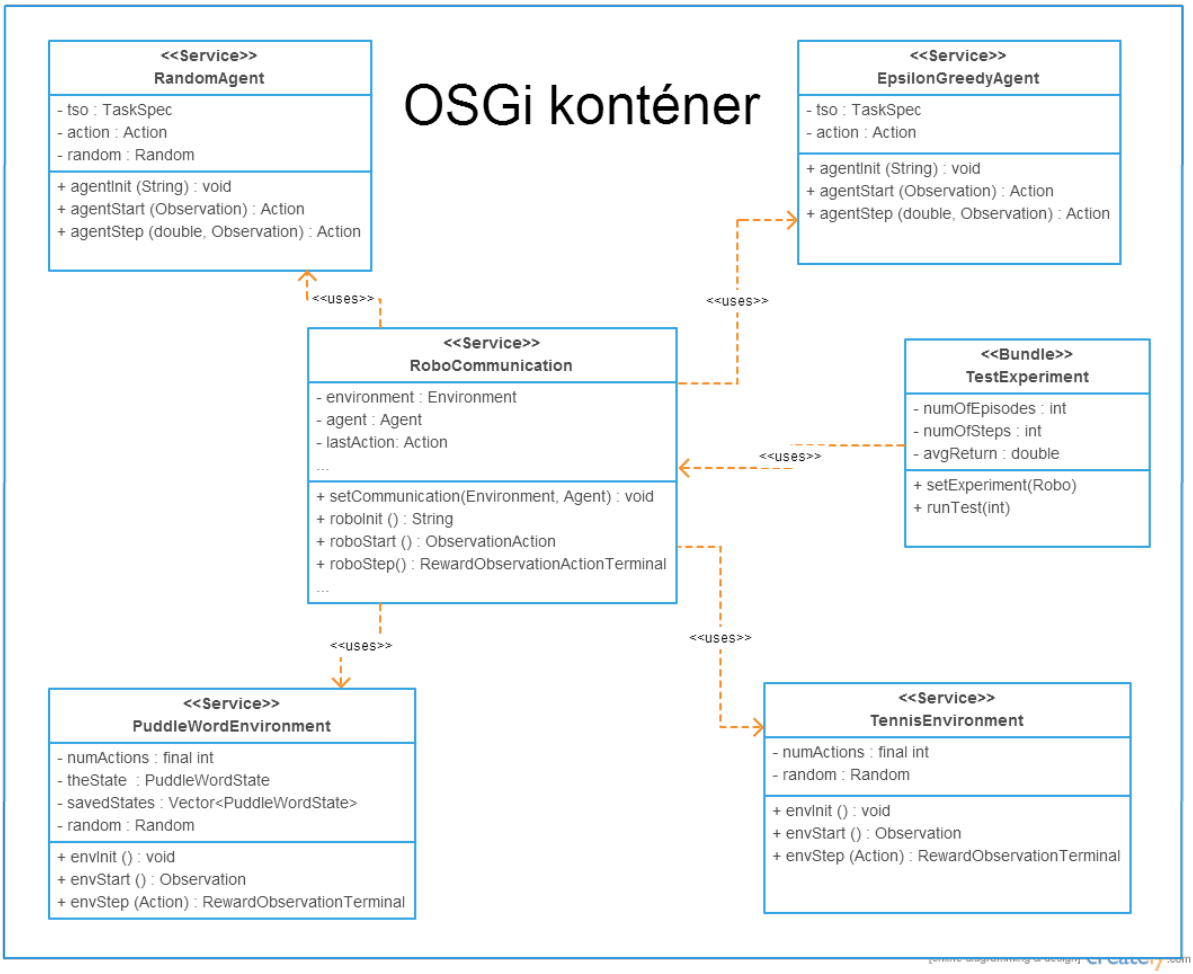
2.1. ábra. Rl-Glue projekt komponensek közti kommunikációja:
http://rl-glue.googlecode.com/svn/trunk/docs/html/index.html

zett 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 `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 szervizeken keresztül így megvalósítva a dinamikus, komponens alapú modell kivitelezését és a több teszt egy időben való futtatási lehetőségét. E mellett ezen szolgáltatások elérése korlátozott, így biztonságot is biztosít ez az architektúra a rendszer számára. Az OSGi konténer futtatásáért egy GlassFish[6] szerver a felelős.

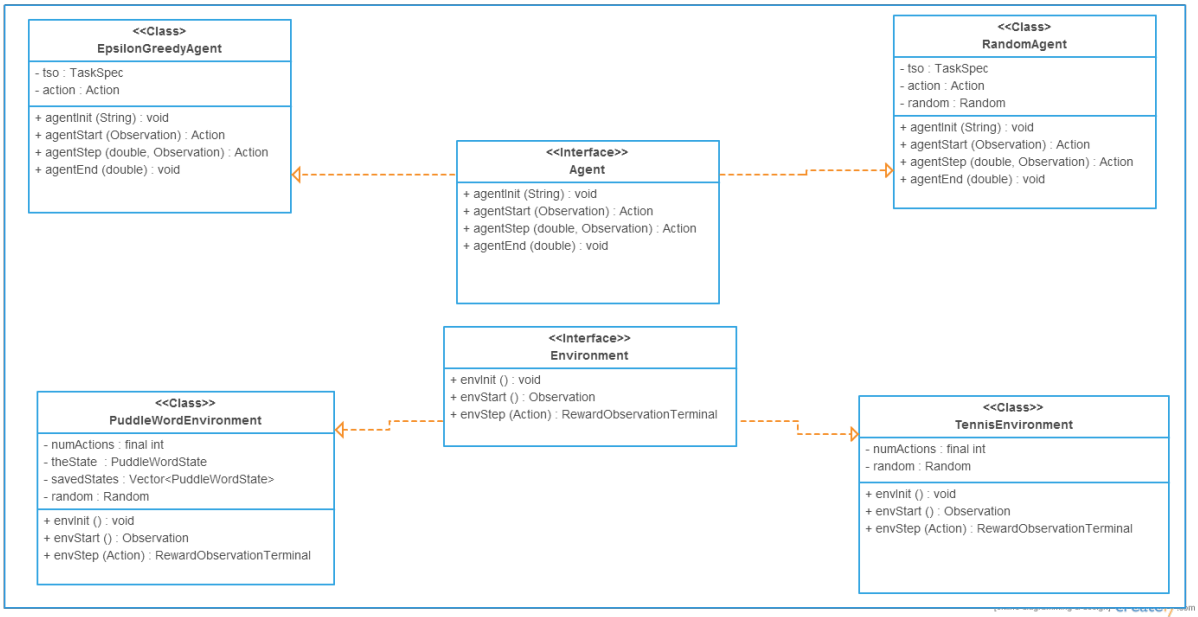
A konténerben egyszerre több előre definiált `Agent` és `Environment` lehet telepítve, ezek száma nincs korlátozva, annyi telepíthető belőlük amennyi még nem okoz gondot a szervert futtató számítógép hardver konfigurációjának. Bármikor módosítható, törölhető vagy teljesen új `Agent` és `Environment` 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` komponenst.



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

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 féltöltésé érdekében, ezekről részletesebben a Rendszer ismertetése!! 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 e három komponens között a folyamatos kommunikációt a teszt futtatása során. A teszt futásának állapotának nyomon követésére lehetőség van a webes felületen keresztül, illetve a teszt befejezésével megtekinthetőek a teszt futása során létrejött adatok halmaza.

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` implementálja az `AgentInterface` -t és minden `Environment` implementálja az `EnvironmentInterface` -t, melyet a 2.3 ábra szemléltet.



2.3. ábra. Agent és Environment Interfész és implementáció

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

Felhasznált technológiák

Összefoglaló: *A fejezet ismerteti a Glassfish alkalmazás szervert és ennek szerepét a RoboRun projekt esetében, illetve a Vaadin technológiát, mely a projekt webes felületének megvalósításában játszott szerepet.*

4.1. Glassfish

A Glassfish alkalmazásszerver a Java Enterprise Edition specifikáció referencia implementációja. Az Oracle tulajdonába tartozik és teljesen nyílt forráskódú, viszont vásárolható hozzá kereskedelmi licensz 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 alkalmazásszerverre telepített OSGi konténerben vannak elhelyezve a különálló komponensek, amelyek együtt alkotják a RoboRun projektet. A Glassfish alkalmazásszerver révén távolról is telepíthetőek könnyedén új komponensek, illetve használhatóak. A Glassfish manageli a RoboRun projekt webes felületét is, mely Vaadin[13] a technológián alapszik.

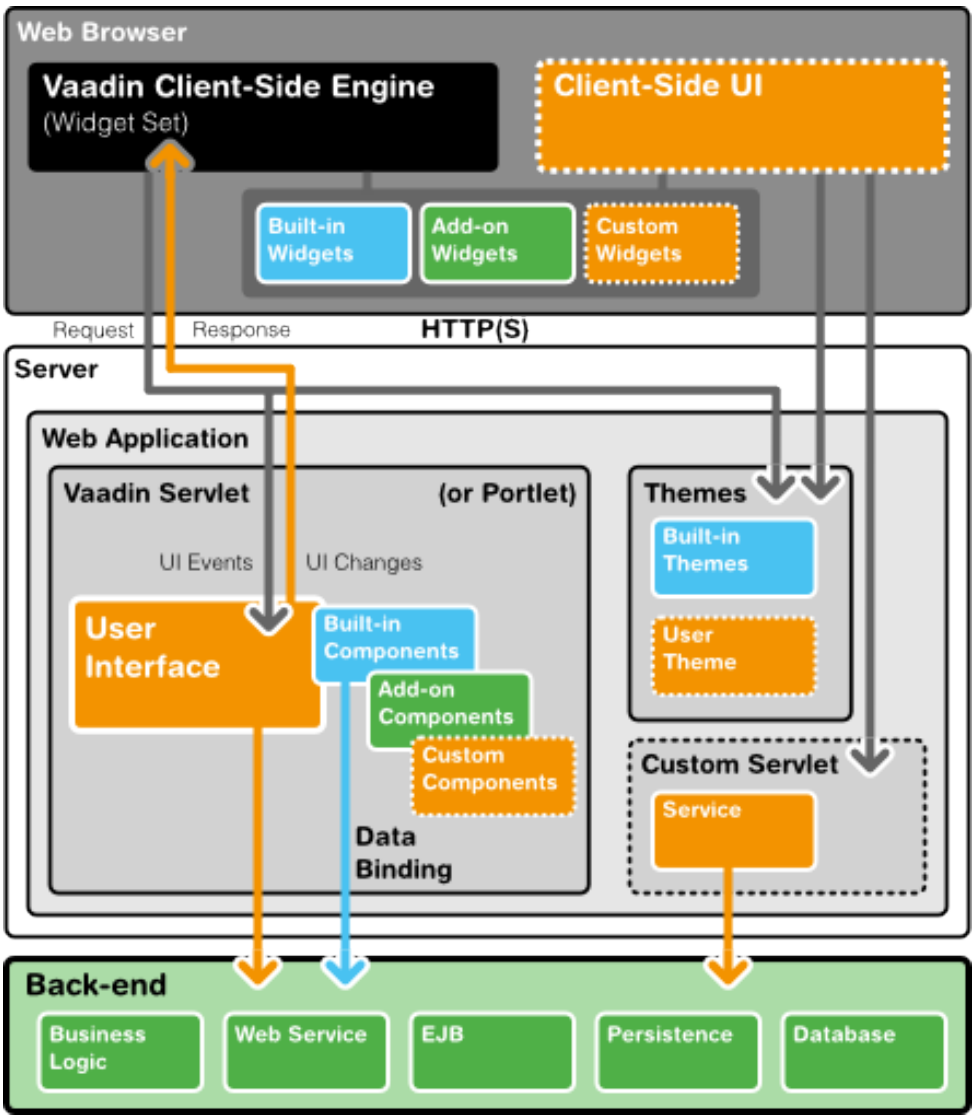
4.2. Vaadin

A Vaadin egy olyan Java webalkalmazás-keretrendszer, amely lehetőséget biztosít gazdag webalkalmazások[1] 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[2] 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, telepíthetőek különböző kiegészítők a még könnyebb használat és a még látványosabb felhasználói élmény érdekében. A komponensek személyre szabhatóak a CSS, HTML5, JavaScript technológiák felhasználása által.

1. http://hu.wikipedia.org/wiki/Rich_Internet_Application
2. http://hu.wikipedia.org/wiki/Google_Web_Toolkit

4.2.1. A Vaadin architektúra

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

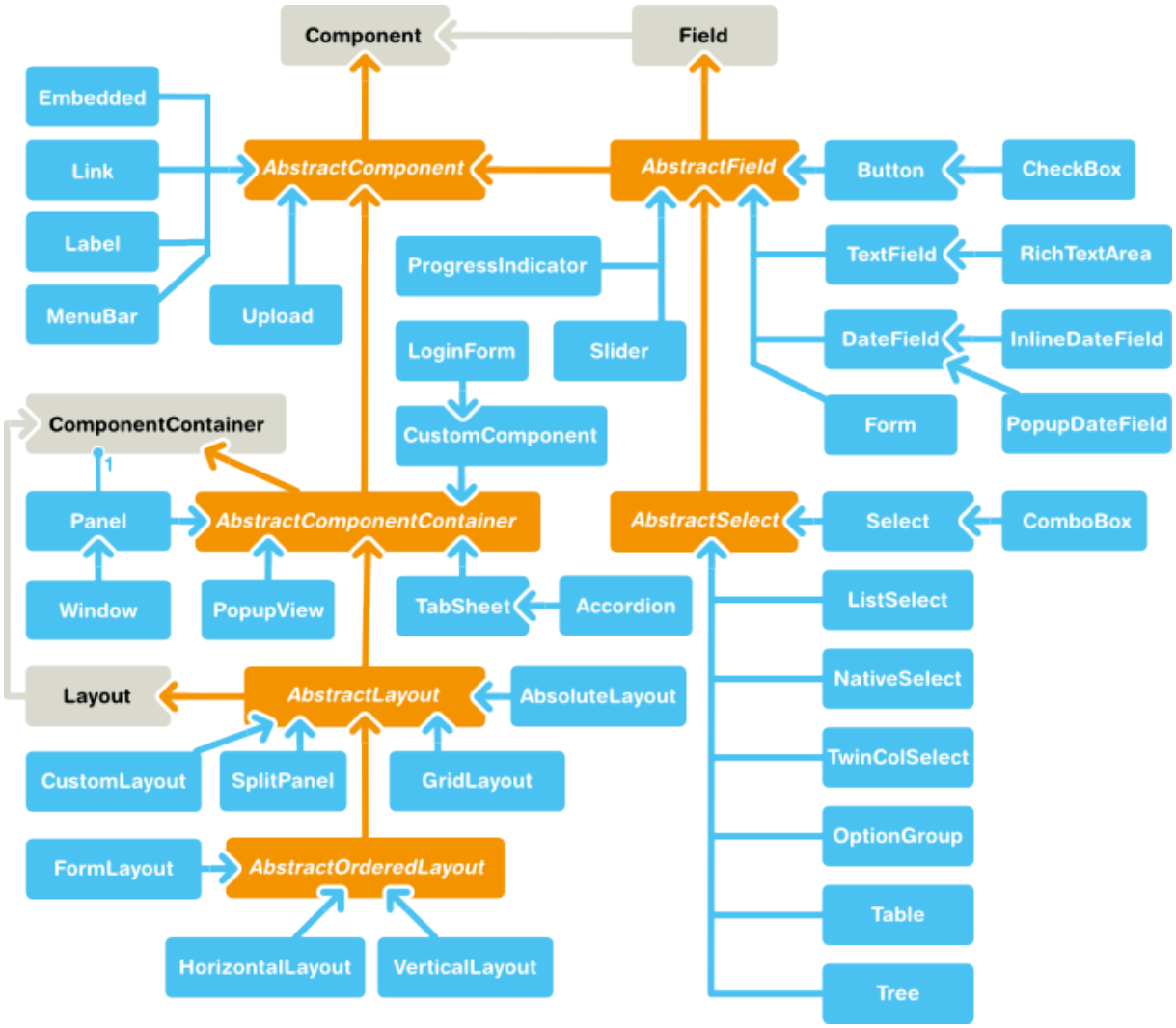


4.1. ábra. Vaadin architektúra https://vaadin.com/book/-/page/architecture.html

A 4.1 ábra szemléleti 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 megjelenítésér felelősek. A szerver felelős az üzleti logika megvalósításáért. A Service réteg valósítja meg a kommunikációt a Back-end réteg és a kliens réteg között.

4.2.2. 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 bővíthetőek, teljesen személyre szabhatóak a CSS, HTML5, JavaScript technológiák által. A 4.1 ábra szemlélteti a Vaadin keretrendszer architektúráját, ahol megfigyelhető, hogy lehetőség van teljesen új komponensek definiálására is. A Vaadin keretrendszer alap komponensei között kapcsolatot, illetve összefüggéseket a 4.2 ábra szemlélteti.



4.2. ábra. Vaadin komponensek
https://inftec.atlassian.net/wiki/display/TEC/Vaadin

A legfelső réteg a `Component` interfész, melyet az `AbstractComponent` absztrakt osztály implementál. Az `AbstractComponent` közös tulajdonságokkal látja el az őt származtató kom-

ponenseket. Az `AbstractComponent` osztályból származtatva van néhány egyszerű komponens, például a `Label`(Címke). A `Component` interfész mellet, megtalálható a Field interfész is, amely örökli a `Component` interfészt. Az `AbstractField` absztrakt osztály 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 `AbstractComponentConainer` osztályból származnak a konténer - alapú komponensek, például a Panel, illetve az elrendezést elősegítő komponensek, például `VerticalLayout`(Függőleges elrendezés).

4.3. További technólógiák

A RoboRun projekt esetén naplózáshoz az SLF4J volt használva. Az SLF4J több naplózási keretrendszer fölött képez absztrakciós szintet, így több különböző naplózási implementációt vehetünk igénybe általa. A RoboRun projekt által igénybe vett implementáció az Apache licenc alatt álló LOG4J.

IDE MÉG ELVILEG BEJÖN A HIBERNATE VAGY JPA.. ESETLEG A JDBC RŐL LEHETNE ÍRNI VALAMIT MEG A MYSQL- RŐL.

IDE IS HA VAN MÉG VALAMI ÖTLETED AZ JÓL JÖN

5. 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.*

5.1. 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.

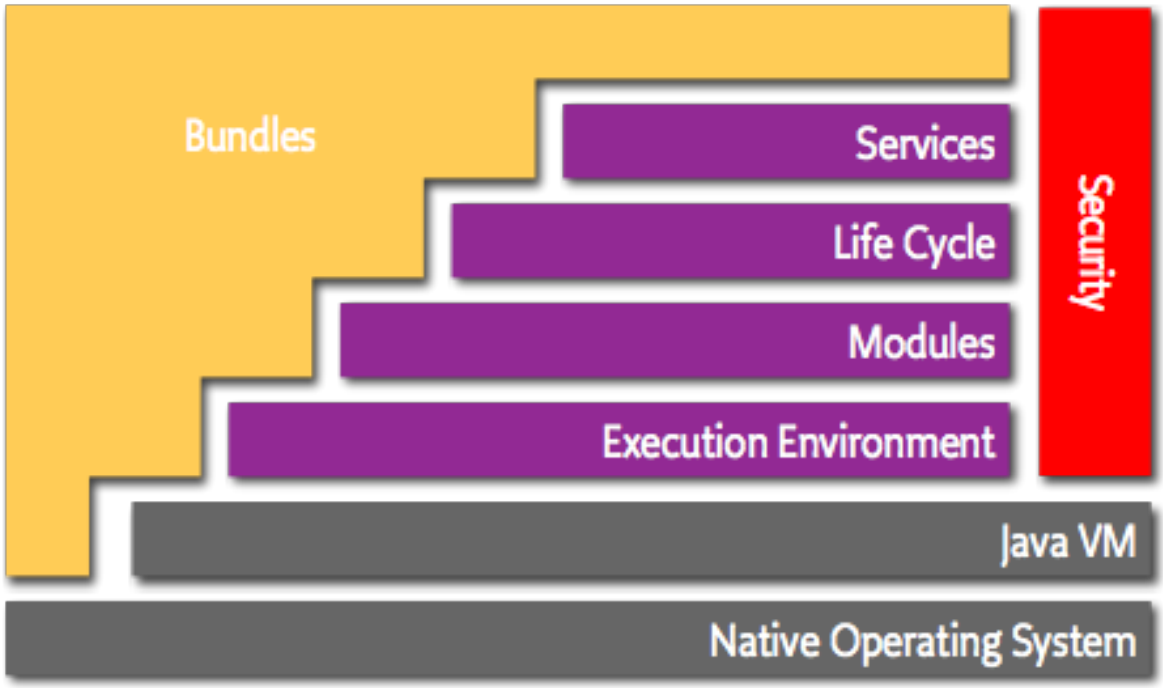
Az OSGi keretrendszer egy olyan keretrendszer, mely a Java nyelv fölött fut. Az OSGi jelentése, Open Service Gateway Initiative. E keretrendszer célja bővíthető Java alkalmazások fejlesztésének a támogatása. Teljesen dinamikus környezetet biztosít, hiszen képes kezelni a csomagok futás idejű megjelenését 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].

5.2. 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



5.1. 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:

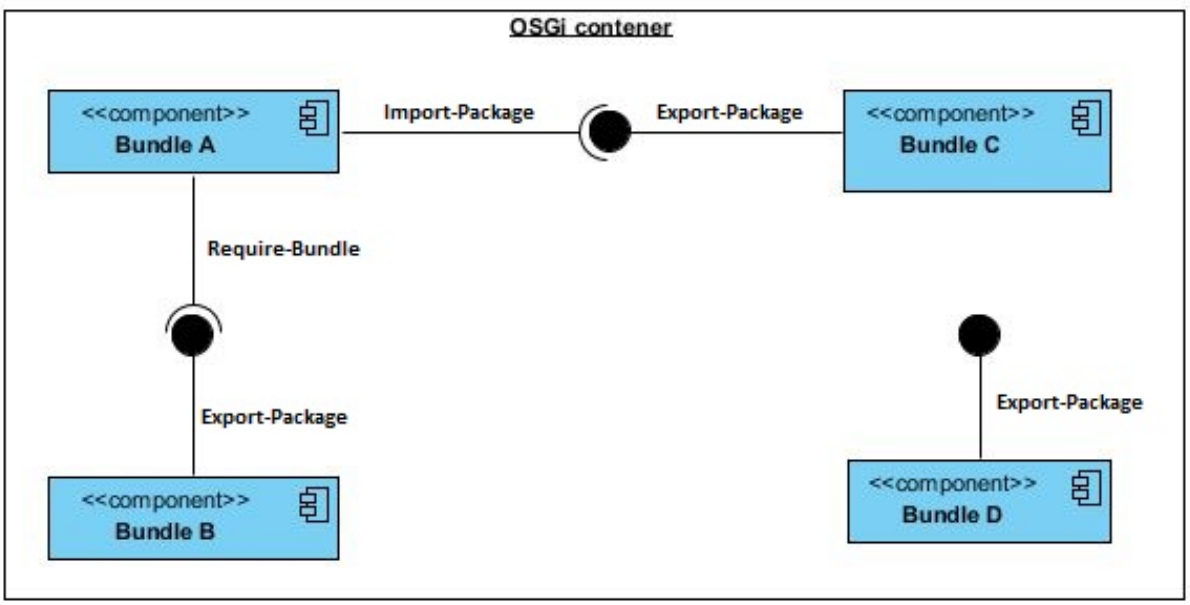
A batyu által létrehozott csomagok

Az **Import-Package** által megadott csomagok

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

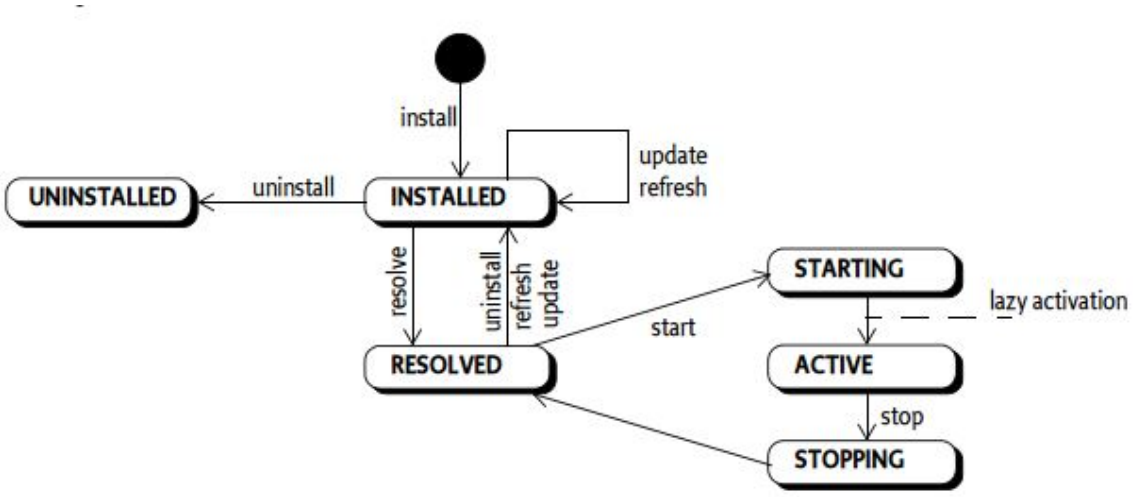
A Java összes függvénykönyvtára

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



5.2. 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.



5.3. 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

5.2.1. 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.

5.3. 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 utolsó sorban könnyen elérhető legyen. Ezen tulajdonságok meghatározása után, a rendszer megvalósításához a legjobb megoldásnak az OSGi keretrendszer 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 keretrendszerben. A Webes felületet biztosító csomag kiterjesztése `war` típusú. A `RandomAgent` szolgáltatás esetén, az alábbi kódrészlet szemlélteti a csomagolási típus megadásának módját:

```
<groupId>edu.bbte</groupId>
<artifactId>agent</artifactId>
<version>0.0.1-SNAPSHOT</version>
<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 szabványnak, miszerint egy projekten belül a `groupID` és `artifactID` párosítások egyediek kell legyenek. A `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.

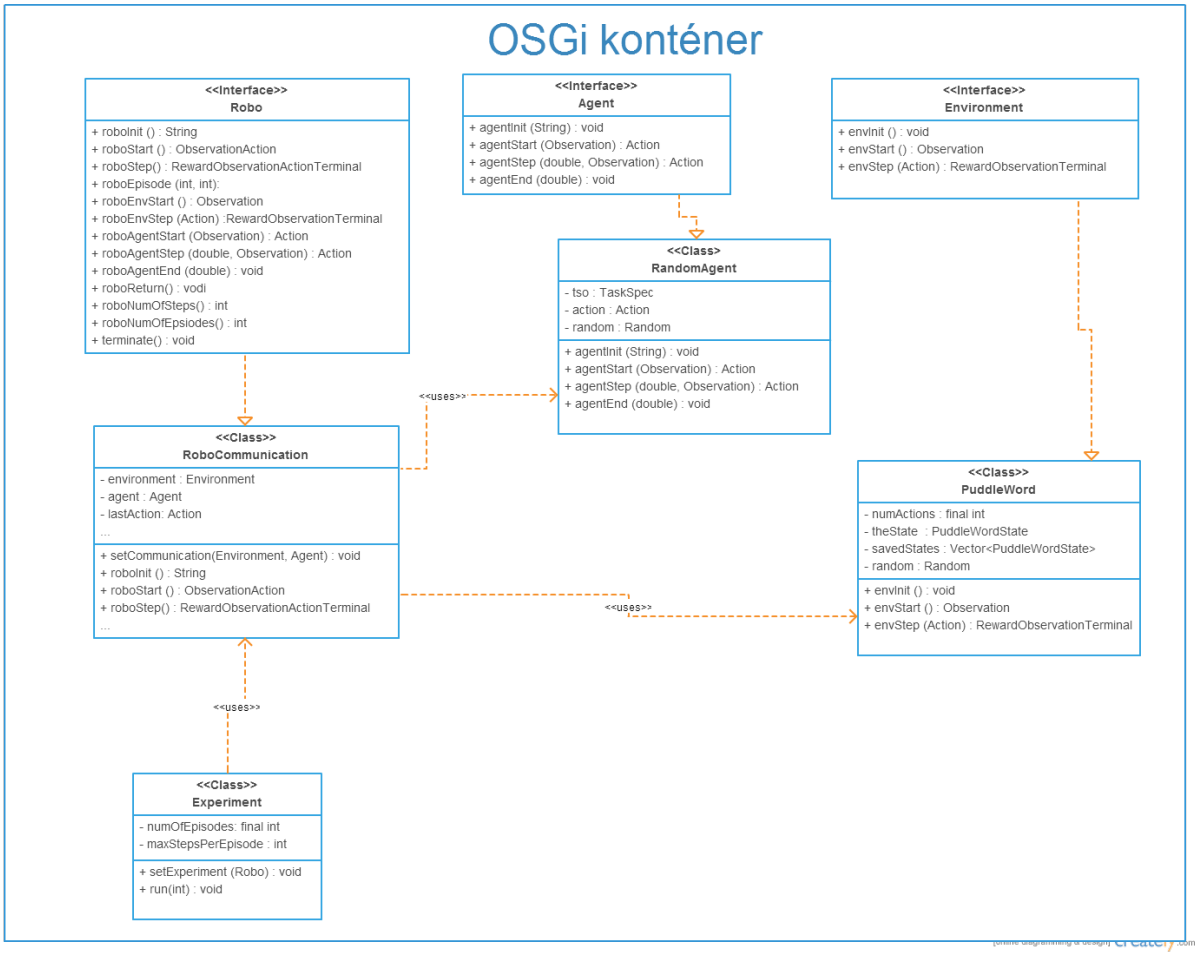
Az `RandoAgent` esetén a függőségek az alábbi kódrészlet alapján alakulnak.

```
        <dependencies>
                <dependency>
                        <groupId>org.osgi</groupId>
                        <artifactId>org.osgi.core</artifactId>
                        <version>4.3.0</version>
                        <scope>provided</scope>
                </dependency>

                <dependency>
                        <groupId>edu.bbte</groupId>
                        <artifactId>packages</artifactId>
                        <version>0.0.1-SNAPSHOT</version>
                </dependency>

                <dependency>
                        <groupId>edu.bbte</groupId>
                        <artifactId>agent</artifactId>
                        <version>0.0.1-SNAPSHOT</version>
                </dependency>
                <dependency>
                        <groupId>edu.bbte</groupId>
                        <artifactId>agentEnvironmentList</artifactId>
                        <version>0.0.1-SNAPSHOT</version>
                </dependency>
        </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 5.3 kódrészlet szemlélteti.

```
<Bundle-SymbolicName>${project.artifactId}</Bundle-SymbolicName>
<Bundle-Version>${project.version}</Bundle-Version>
<Bundle-Activator>edu.bbte.agent.AgentActivator</Bundle-Activator>
<Export-Package>
        edu.bbte.agent;version=${project.version}
</Export-Package>
<Import-Package>
        edu.bbte.packages.types, *
</Import-Package>
```

Az `Agent` interfész által exportált csomag a `edu.bbte.agent` csomag és az által 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.



5.4. ábra. Osgi Alapkomponensek

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 alapkomponensre épül a rendszer. Ezen komponensek az OSGi szabványnak megfelelően, külön szolgáltatások, kivéve az Experiment, mely nem 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 5.4 ábra szemlélteti.

A `Robo`, az `Agent` és az `Environment` interfészek külön batyuk, amelyek exportálják a csomagjukat, 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 kérdezi 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 szolgálta-

tásokkal szeretne dolgozni. Ezen komponensek interakciója révén képes a rendszer megerősítéses tanulási algoritmusok futtatására és tesztelésére.

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 megadott `Agent` és `Environemnt` 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 `Exerpiment` komponensben a következő kódrészlet szemlélteti.

```
1       agentServiceReference                   =
                    context.getServiceReferences(Agent.class.getName(), "(AgentName=
                        RandomAgent)");
        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 megkülönböztethetőek 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 keretrendszerben 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 eredmé-

nyekhez vezet. E probléma kiküszöbölése érdekében, került használatra a `ServiceFactory` interfész, amely az OSGi keretrendszer 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 5.3 szemlélteti.

```
public class RandomAgentServiceFactory implements ServiceFactory {

    private static final Logger logger = Logger.getLogger(RandomAgentServiceFactory.class.
        getSimpleName());
    private int usageCounter = 0;

    public Object getService(Bundle bundle, ServiceRegistration registration) {

        usageCounter++;

        logger.log(Level.INFO,"Create␣object␣of␣RandomAgent␣for␣" + bundle.getSymbolicName());
        logger.log(Level.INFO,"Number␣of␣bundles␣using␣service␣" + usageCounter);

        Agent agent = new RandomAgent();

        return agent;
    }

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

        usageCounter--;

        logger.log(Level.INFO,"Release␣object␣of␣RandomAgent␣for␣" + bundle.getSymbolicName())
            ;
        logger.log(Level.INFO,"Number␣of␣bundles␣using␣service␣" + usageCounter;
    }
}
```

6. fejezet

A rendszer felépítése és használata

Összefoglaló: *E fejezet célja részletesen ismertetni a rendszer teljes architektúrájának felépítését és a fejlesztés során felmerülő problémák is kiemelésre kerülnek. A fejezet második részében a rendszer használatának ismertetése található.*

6.1. Specifikáció

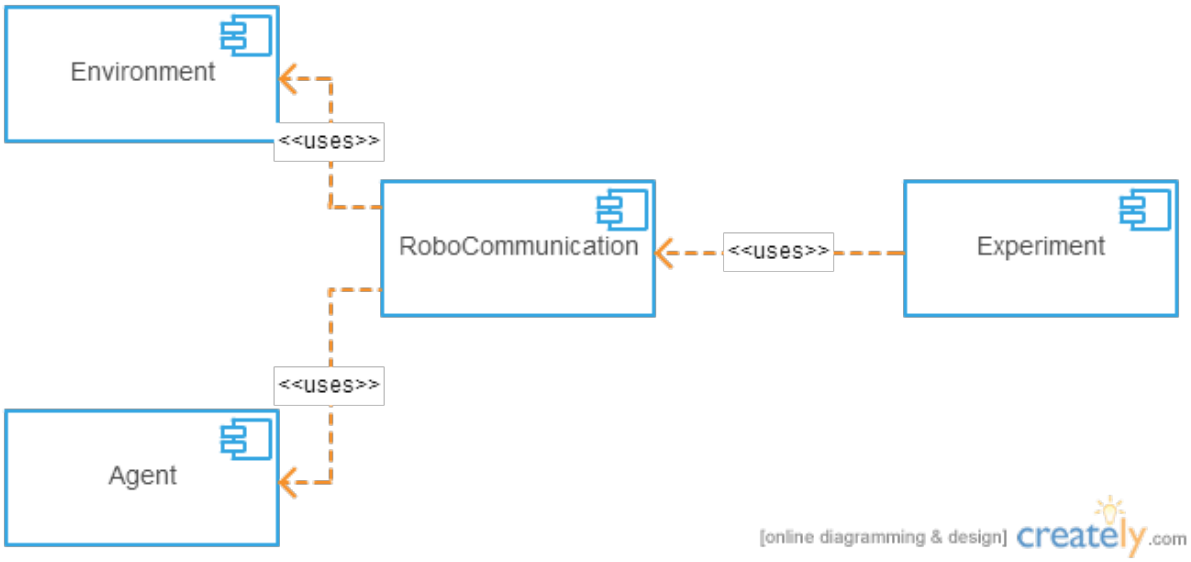
A RoboRun projekt alapelképzelése egy olyan szoftver megalkotása, amely teljesen önállóan képes futtatni megerősítéses tanulással kapcsolatos algoritmusokat. Ezen algoritmusok futtatása, egy távoli szervergépen történik, amely folyamatosan elérhető. A tesztek futtatása szempontjából fontos a gyorsaság 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. Megtekinthetőek az aktív tesztek állapotai és különböző fontos információk az aktív tesztekről, például: indítás ideje, aktuálisan hol tart, milyen `Agent` illetve `Environment` példánnyal fut a teszt. A webes felületen megtekinthetőek a már befejeződött tesztek által generált adatok illetve statisztikai a hozzájuk tartozó statisztika.

6.2. A rendszer felépítése

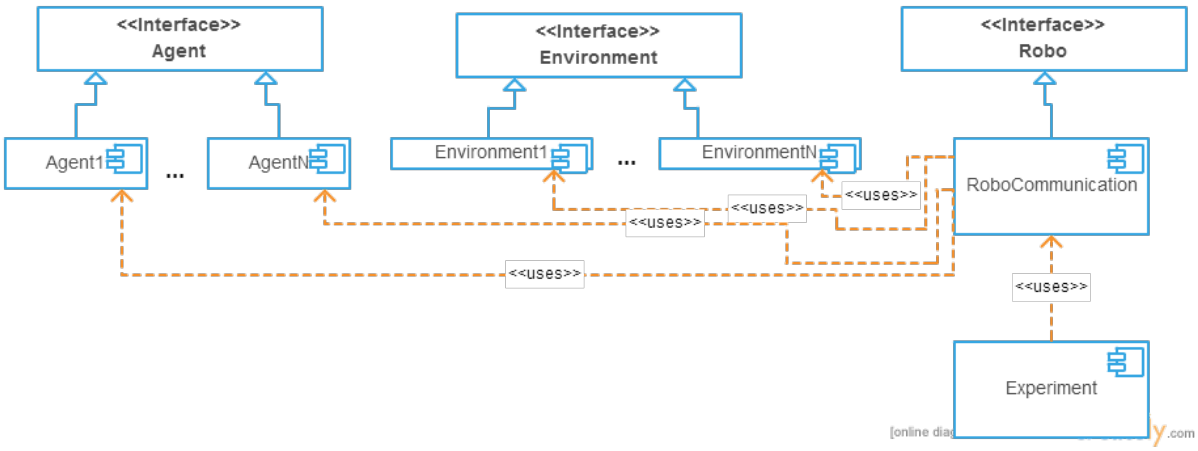
A RoboRun projekt négy alapkomponensre épül. Melyek az `Agent`, `Environment`, `Experiment` és `RoboCommunication` komponensek formájában vannak megvalósítva. Az `Agent` komponens valósítja meg a mesterséges tanulási algoritmusokat. Az `Environment` a környezet, amelyet az `Agent` próbál megtanulni helyesen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

komponens felelős az előbbi három komponens között teljes kommunikáció lebonyolításáért egy teszt során, melyet a 6.1 ábra szemléltet.



6.1. ábra. A RoboRun projekt Alap komponensei

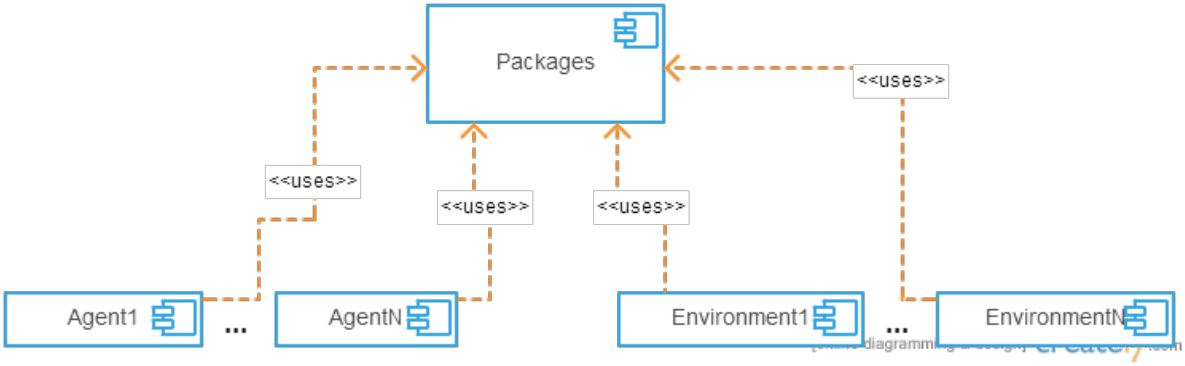
Az `Experiment` komponens kivételével, minden alapkomponens egy neki megfelelő interfészt implementál. Ez által kerül megvalósításra az, hogy a rendszerben egyszerre több `Agent` és `Environment` példány lehet telepítve, illetve ezek cseréje és módosítása is lényegesen könnyebb. A `RoboCommunication` komponensből egy található a rendszerben, viszont ez is könnyedén módosítható vagy cserélhető anélkül, hogy az egyéb komponenseket módosítani kellene. A RoboRun projekt esetén az alapkomponensek és a hozzájuk tartozó interfészeket a 6.2ábra szemlélteti.



6.2. ábra. A RoboRun projekt alapkomponensei és interfészek

Az alapkomponenseknek szükségük van néhány saját típusra a tesztek futtatásához. Ezen típusok egy külön batyuban kaptak helyet, melynek neve `packages`. Ezen típusok az Rl-Glue projektből lettek átvéve és felhasználva módosítás nélkül. A `packages` batyu közzéteszi a különböző csomagjait, amelyet

az alapkomponensek 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.



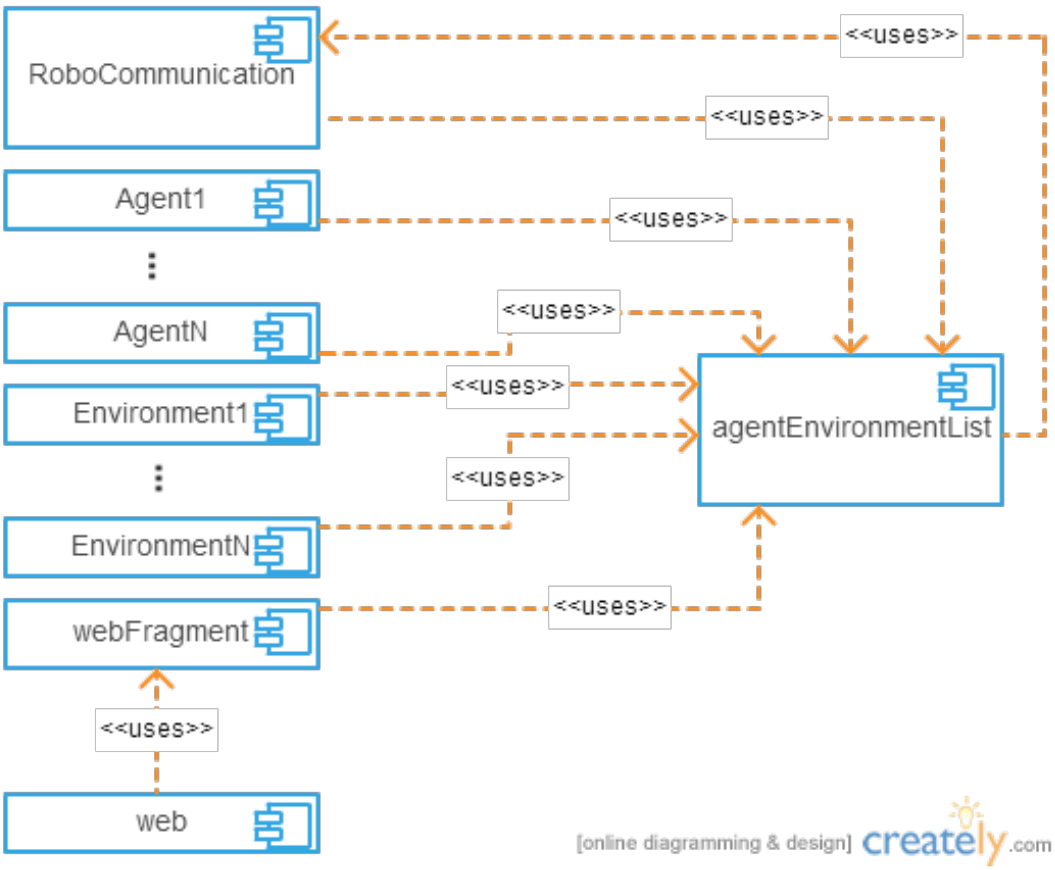
6.3. ábra. Az alapkomponensek által felhasznál komponens

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 tesztekről. A fájlok a webes felület segítségével érhetőek el. A `Results` végződésű fájl letölthető a nagy adatmennyiséget figyelembe véve. A statisztikai adatok megtekinthetőek a webes felületen. A 6.4 ábra szemlélteti a kapcsolatot az adat hozzáférés réteg és a `RoboCommunication` komponensek között.

6.4. ábra. Az adathozzáférési réteg és a RoboCommunication

A webes megjelenítéshez szükség van egy olyan komponensre, amely összegyűjti az adatokat a többi komponenstől 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 végrehajtódjon 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` komponenstől és ezt átadja a `web` komponensnek megjelenítésre. Ezen komponensek kapcsolatáról a 6.5 ábra nyújt összefogóbb képet.



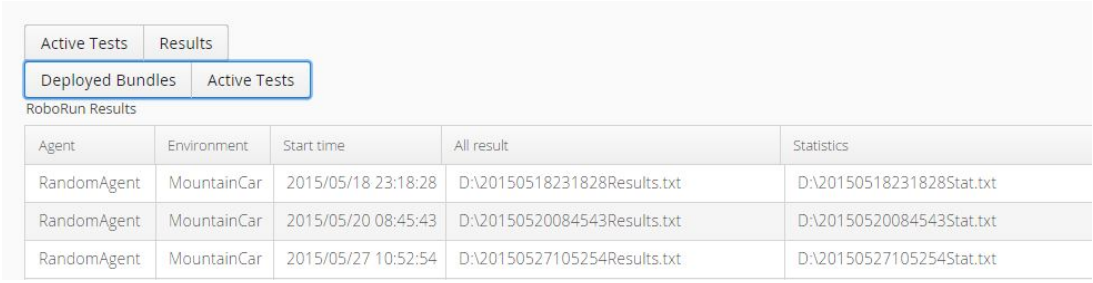
6.5. ábra. A webes felület megjelenítéséért felelős komponensek

6.2.1. 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 komponenst 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 könnyed továbbfejlesztést. Ahhoz, hogy a fragmentek könnyedén telepíthetőek legyenek, szükség van használni az OSGi keretrendszer által kínált `SeviceTracker` osztályt, mely a `ServiceTrackerCustomizer` interfészt implementálja. Három metódussal rendelkezik az interfész. Az `addinService()` metódussal mely egy szolgáltatás referenciát vár paraméterként, illetve 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 funkcionalitások.

6.2.2. Webes felület funkcionalításai

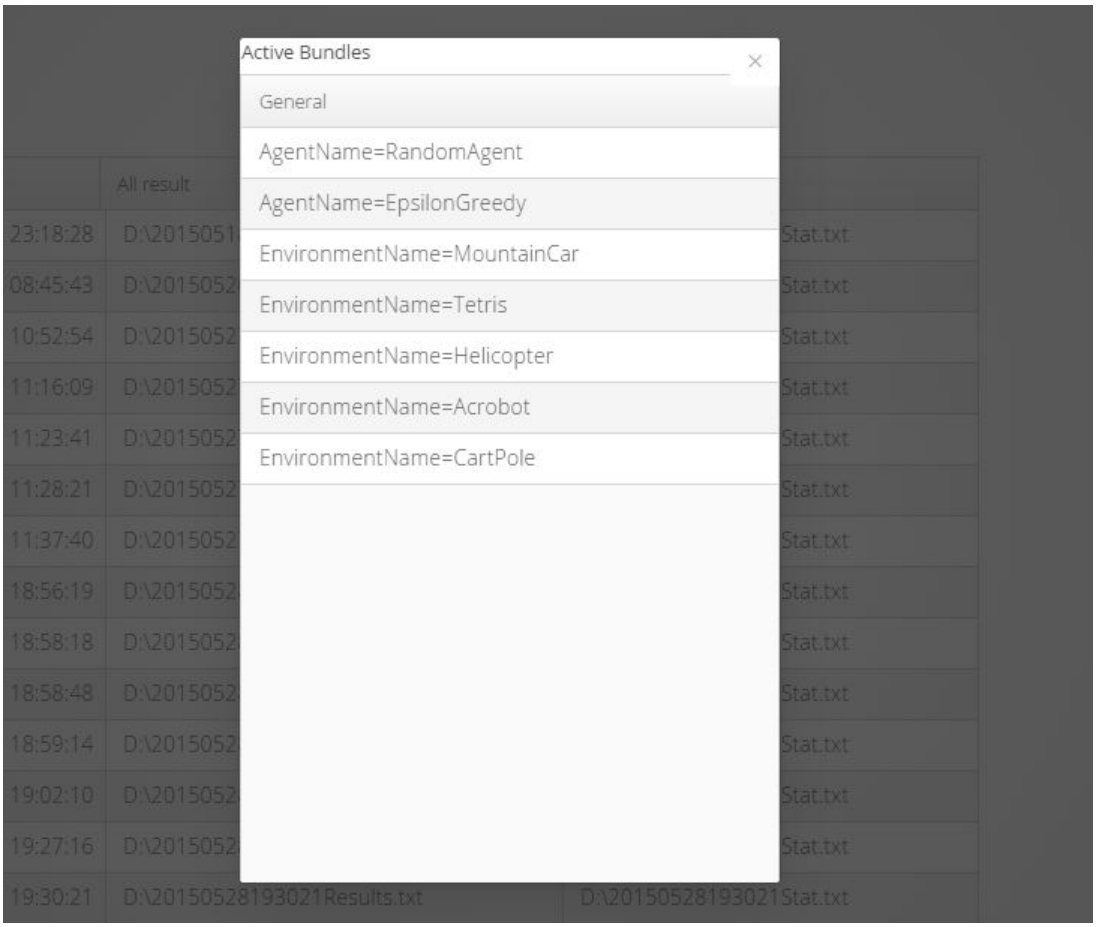
Az oldal betöltődését követően a felhasználó az aktív tesztek állapotairól szerezhet tudomást. Az aktív tesztekről látható, hogy melyik `Agent` és `Environment` példánnyal dolgozik az adott teszt. Látható a teszt indításának ideje és az, hogy a teszt éppen hol tart.



6.6. ábra. Aktív tesztek nézet

A 6.6 kép szemlélteti ezt a nézetet.

Az `Agent and Environment List` 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. Ezt a 6.7 kép szemlélteti.



6.7. ábra. Aktív Agent és Environmentek nézet

Az tesztek összesítő adatait illetve a statisztikai adatokat a `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. Ez utóbbi megtekinthető a webes felületen, előbbi viszont letöltést igényel a nagy adatmennyiség miatt. E nézetet a 6.8 kép szemlélteti.

6.3. 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

RoboRun Results

Agent	Environment	Start time	All result	Statistics
RandomAgent	MountainCar	2015/05/18 23:18:28	D:\20150518231828Results.txt	D:\20150518231828Stat.txt
RandomAgent	MountainCar	2015/05/20 08:45:43	D:\20150520084543Results.txt	D:\20150520084543Stat.txt
RandomAgent	MountainCar	2015/05/27 10:52:54	D:\20150527105254Results.txt	D:\20150527105254Stat.txt
RandomAgent	MountainCar	2015/05/27 11:16:09	D:\20150527111609Results.txt	D:\20150527111609Stat.txt
RandomAgent	MountainCar	2015/05/27 11:23:41	D:\20150527112341Results.txt	D:\20150527112341Stat.txt
RandomAgent	MountainCar	2015/05/27 11:28:21	D:\20150527112821Results.txt	D:\20150527112821Stat.txt
RandomAgent	MountainCar	2015/05/27 11:37:40	D:\20150527113740Results.txt	D:\20150527113740Stat.txt
RandomAgent	MountainCar	2015/05/28 18:56:19	D:\20150528185619Results.txt	D:\20150528185619Stat.txt
RandomAgent	MountainCar	2015/05/28 18:58:18	D:\20150528185818Results.txt	D:\20150528185818Stat.txt
RandomAgent	MountainCar	2015/05/28 18:58:48	D:\20150528185848Results.txt	D:\20150528185848Stat.txt
RandomAgent	MountainCar	2015/05/28 18:59:14	D:\20150528185914Results.txt	D:\20150528185914Stat.txt
RandomAgent	MountainCar	2015/05/28 19:02:10	D:\20150528190210Results.txt	D:\20150528190210Stat.txt

6.8. ábra. A teszt eredményeket tartalmazó nézet

- `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 funkcionalitásokat
- `RoboCommunication` - amely az egyes `Agent` és `Environment` példányok közötti kommunikációért felelős

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

6.4. Tesztek futtatásának folyamata

Egy teszt futtatásának folyamata

IDE EGY FOLYAMAT ÁBRA

7. 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.

EZ ELÉGGÉ VÁZLAT JELLEGŰ, EZT MÉG KI KELL EGÉSZÍTENEM! ESETLEG HA VAN ÖTLETED, HOGY MIKET LEHET IDE ÍRNI AZ JÓ LENNE!

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.

[13] Vaadin hivatalos weboldal. https://vaadin.com/home. Utolsó megtekintés dátuma: 2015-05-30.