Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Srovnání prostředí pro modelování SW agentů

Plzeň, 2013 Vladimír Zeman

PROHLÁŠENÍ

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 7. května 2013

Vladimír Zeman

PODĚKOVÁNÍ

Rád bych touto cestou vyjádřil poděkování Ing. Richardovi Lipkovi, Ph.D. za vstřícný přístup, odborné vedení, trpělivost a podnětné návrhy k mé bakalářské práci.

ABSTRACT

Comparsion of tools for modeling of software agents

The object of this work is to compare contemporary tools for modelling software agents. It presents eleven different tools, features their brief characteristics and focuses closer on the tools MASON and Repast Simphony 2.0, which are mainly used for creating simulations. The work also includes information about installation and basic manuals for working with these tools, respectively instructions for creating a project and run the sample simulations.

ABSTRAKT

Předmětem této práce je srovnat současné nástroje pro modelování softwarových agentů. Představuje jedenáct různých nástrojů a uvádí jejich stručnou charakteristiku. Blíže se zaměřuje na nástroje MASON a Repast Simphony 2.0, které se využívají převážně pro vytváření simulací. Součástí práce jsou i informace o instalaci a základní manuály pro práci s těmito nástroji, resp. pokyny pro vytvoření projektu a spouštění ukázkových simulací.

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

1 ÚVOD

Problematika agentů a multiagentních systémů se v současné době stává jednou z dominantních oblastí výzkumu umělé inteligence. Postupně se vymezuje jako samostatná disciplína vycházející z distribuované umělé inteligence. Jako taková se opírá o výsledky výzkumů v oblasti počítačových věd, jiných disciplín umělé inteligence, ale také o poznatky z celé řady dalších vědních oborů jako např. sociologie, ekonomie, psychologie a biologie.

Se softwarem založeném na agentech se můžeme setkat v mnoha oblastech počínaje výukovými programy, simulacemi, počítačovými hrami, roboty pro prohledávání, nakupování a prodej na internetu, až po rozsáhlé distribuované průmyslové systémy.

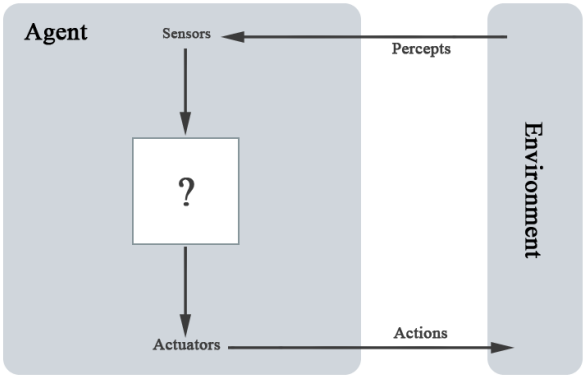
Bakalářská práce je zaměřena na srovnání prostředí pro modelování softwarových agentů. V úvodní části je vysvětleno, co se skrývá pod pojmem agent, jak lze agenty dělit, a v jakých prostředích se mohou agenti nacházet. V další části je představeno jedenáct různých nástrojů a prostředí pro vytváření softwarových agentů. U každého z nich je uvedena stručná charakteristika a příklady použití.

Další prostor je věnován dvěma vybraným nástrojům, kterými jsou MASON a Repast Simphony 2.0. Tato část má čtenářům blíže popsat zmiňované nástroje a usnadnit jim vytváření a spouštění prvních modelů simulací. Poslední část se zabývá srovnáním těchto nástrojů na základě vlastních zkušeností s vytvořením dvou modelů simulací v obou těchto prostředích.

2 CHARAKTERISTIKY A TYPY AGENTŮ

2.1 Agent

Agent může být chápán jako něco vnímající své prostředí skrze senzory a působící na toto prostředí skrze aktuátory. Tento jednoduchý princip je vidět na obr. 2.1. V případě softwarových agentů můžeme za senzorické vstupy považovat stisknutí klávesy, obsahy souborů nebo síťové pakety. Působení na okolí může mít formu výstupu na obrazovku, zápisů do souboru a odesílání síťových paketů. Předpokládá se, že každý agent vnímá své vlastní akce (ne ale vždy důsledky) [7].



Obr. 2.1: Ovlivňování agenta s prostředím skrze senzory a aktuátory [7].

O agentovi obecně platí, že:

- může posílat a přijímat informace od jiných agentů za použití vhodných protokolů;
- může zpracovávat přijaté informace a uvažovat o nich (tj. provádět odvozování, syntézu i analýzu);
- má soubor schopností provádět akce, které se mohou i dynamicky měnit (tj. akce charakterizují úkoly, které dokáže agent provádět).

Inteligentní agent je entita, která je zodpovědná za rozhodování, zda a jak reagovat na externí podněty.

Inteligentní agent má schopnost plnit cíle s využitím logické dedukce, tj. navíc umí:

- uvažovat o svých schopnostech a schopnostech ostatních agentů;
- generovat cíle nebo plány pro sebe a jiné agenty;
- účastnit se složitých interakcí s ostatními (např. za účelem vyjednávání a delegování úkolů), dynamicky se zapojovat do skupin či organizací, které mohou například sdružovat agenty s podobnou funkcí, nebo také skupiny opouštět;
- získávat informace a používat jejich zdroje a udržovat explicitní modely důvěry pro sebe a ostatní agenty [6].

2.2 Struktura agenta

agent = architektura + program

- **architektura** - výpočtové zařízení s fyzickými senzory a aktuátory;
- **program** - implementace agentovy funkce;
  - mapuje vjemy na akce;
  - přesněji řečeno, vstupem je jeden vjem (jediné, co je přímo dostupné z prostředí prostřednictvím čidel);
  - pokud akce záleží na posloupnosti vjemů, musí si ji agent sám pamatovat;
  - samozřejmě program musí být vhodný pro danou architekturu [17]!

2.3 Základní charakteristiky agenta

Agenta lze definovat jako hardwarový nebo častěji softwarový počítačový systém, který má tyto základní charakteristiky [1]:

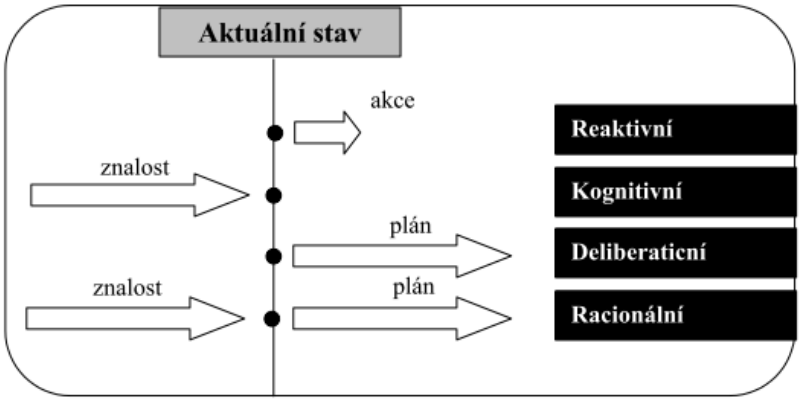
- **autonomie** - agenti pracují bez přímého zásahu člověka nebo jiných agentů a mají kontrolu nad svými akcemi a vnitřním stavem;
- **sociální schopnosti** - agenti se vzájemně ovlivňují s jinými agenty (možná i lidmi) pomocí jazyka určeného pro komunikaci mezi agenty;
- **reaktivita** - agenti vnímají své prostředí (což může být fyzický svět, uživatel pomocí grafického uživatelského prostředí, skupina jiných agentů, internet nebo kombinace všeho) a reagují včas na změny, které v něm nastanou;
- **proaktivita** - agenti nejednají pouze v reakci na své prostředí, jsou schopni vykazovat cílené chování a sami přebrat iniciativu.

2.4 Typy agentů

Agenty lze rozlišit na následující typy [6]:

- **reaktivní agent** – bezprostředně reaguje na jisté změny prostředí (nebo své změny vůči prostředí), aniž by měl vnitřní reprezentaci znalostí o tomto prostředí. Jeho reakce nejsou výsledkem výpočtů či dedukcí na základě znalostí, ale pouze reakcemi na podněty;
- **deliberativní agent** – na rozdíl od reaktivního agenta má deliberativní (rozvážný) agent schopnost plánovat postup svých akcí vedoucích k dosažení zvolených nebo zadaných záměrů/cílů. To znamená, že agent musí mít schopnost různých výpočtů. K dosažení svých záměrů pak agent ovlivňuje okolní prostředí tak, aby získal nějakou výhodu;

- **kognitivní agent** – má schopnost vyvozovat logické závěry ze svých pozorování okolního prostředí. Takový agent musí být schopen se především učit a vytvářet si svou vlastní bázi znalostí. Do ní si během svého působení ukládá informace získané interakcí s okolím nebo znalosti získané dedukcí. Kognitivní agent nemusí mít nutně deliberativní schopnosti. Pak provádí pouze vnitřní akce, například analyzuje scénu, provádí překlad, nebo získává znalosti (dolování znalostí z dat);
- **racionální agent** - má všechny výše uvedené vlastnosti a jeho struktura obsahuje jak plánovací jednotku, tak i kognitivní jednotku včetně báze znalostí. Je to agent, který je na základě svých poznatků schopen se učit a pak plánovat svoji činnost tak, aby dosáhl svých cílů racionálním způsobem. Stojí nejvýše na pomyslné hierarchii uvedených agentů (viz obr. 2.2).



Obr. 2.2: Základní rozdělení agentů [6].

Za speciální třídu agentů můžeme považovat **mobilní agenty**. Mobilní agent je program, který je schopen autonomně (bez zásahu člověka) přesouvat svůj programový kód z počítače na počítač a pokračovat v jeho vykonávání. Podmínkou je existence prostředí pro jeho správu (vytvoření agenta, jeho kompilaci nebo interpretaci, přesun a ukončení jeho procesu). Základní výhodou mobilních agentů ve srovnání s agenty stacionárními je přenos funkcí k datům a šetření zdrojů komunikační infrastruktury.

Mezi nejvýznamnější výhody mobilních agentů patří:

- *Snížení zátěže v komunikační síti* - operabilita agentů na místě zpracování dat umožňuje snížit jak množství přenášených dat, tak i náklady na komunikaci;
- *Zlepšení adaptivity agentů na změny v prostředí* - mobilní agenti přizpůsobivě reagují na změny svého prostředí, což zvyšuje jejich robustnost. V případě potřeby se mohou transportovat na jiný počítač;

- *Heterogennost* - mobilní agenti jsou zpravidla nezávislí na hardwarové platformě a závisí jenom na agentovém systému, kterému přináleží.

2.5 Prostředí agentů

Pojmem prostředí je v kontextu agentních systémů chápán veškerý okolní svět, se kterým agent přichází během své činnosti do styku. V tabulce 2.1 je uvedena řada známých prostředí spolu s jejich vlastnostmi, které lze popsat následovně [7]:

- **plně pozorovatelné / částečně pozorovatelné**
  - *plně pozorovatelné prostředí* - senzory agenta mají přístup ke kompletnímu stavu prostředí v jakémkoli okamžiku;
  - *částečně pozorovatelné prostředí* - agent není schopen svými senzory získat kompletní stav prostředí;
- **deterministické / nedeterministické**
  - *deterministické prostředí* - následující stav prostředí je plně určen současným stavem a akcí vykonanou agentem;
  - *strategické prostředí* - jedná se o deterministické prostředí, ve kterém stav prostředí mění i další agenti;
  - *nedeterministické prostředí* - následující stav prostředí nelze dopředu predikovat;
- **epizodické / sekvenční**
  - *epizodické prostředí* - zkušenosti agenta se dělí na atomické epizody. V každé epizodě agent vnímá své okolí a pak provádí určitou akci bez ohledu na akce provedené v předchozích epizodách (akce závisí pouze na dané epizodě);
  - *sekvenční prostředí* - současné rozhodnutí může ovlivnit všechna budoucí rozhodnutí;
- **statické / dynamické**
  - *statické prostředí* - nemění se, zatímco agent přemýšlí;
  - *dynamické prostředí* - může se měnit bez zásahu agenta;
  - *semi-dynamické* - prostředí se s časem nemění, mění se ale míra výkonu agenta;
- **diskrétní / spojité**
  - *diskrétní prostředí* - má konečný nebo spočetný počet stavů;
  - *spojité prostředí* - má nekonečný počet stavů;
- **jedno agentní / multiagentní**
  - *jedno agentní prostředí* - v prostředí se vyskytuje pouze jeden agent, případně více nezávislých agentů s vlastními cíli, akcemi a znalostmi;
  - *multiagentní prostředí* - lze dále dělit na:

* *kompetitivní multi-agentní prostředí* - agenti mají protichůdné cíle;
* *kooperativní multi-agentní prostředí* - agenti mají společné cíle.

Tab. 2.1: Příklady prostředí a jejich charakteristik

Úloha prostředí	Pozoro-vatelnost	Deterministické	Epizodické	Statické	Diskrétní	Agenti
Křížovka	Úplná	Deterministické	Sekvenční	Statické	Diskrétní	Jeden
Šachy s hodinami	Úplná	Strategické	Sekvenční	Semi	Diskrétní	Multi
Poker	Částečná	Stochastické	Sekvenční	Statické	Diskrétní	Multi
Backgammon	Úplná	Stochastické	Sekvenční	Statické	Diskrétní	Multi
Řízení taxi	Částečná	Stochastické	Sekvenční	Dynamické	Spojité	Multi
Analýza obrázku	Částečná	Deterministické	Epizodické	Semi	Spojité	Jeden

3 NÁSTROJE PRO TVORBU SW AGENTŮ

3.1 ZEUS

ZEUS poskytuje knihovny softwarových součástí a nástrojů, které usnadňují rychlý návrh, vývoj a zavádění agentních systémů. Skládá ze tří sad hlavních nástrojů [3]:

- *Agent Component Library* - sada softwarových komponent, které implementují funkce nezbytné pro multiagentní systémy;
- *Agent Building Tools* - integrované grafické nástroje, které programátora provedou všemi etapami vývoje komplexního agenta, během kterých nastaví způsob interakce mezi agenty a jaké úkoly bude agent plnit v rámci své aplikace. Nastavení se provádí pomocí strukturovaného menu a tabulek. Mezi tyto nástroje patří:
  - *Ontology Editor* - definování konceptů, vlastností a omezení v rámci domény;
  - *Agent Definition Editor* - popis agentovy logiky (úkoly, počáteční zdroje, plánovací schopnosti, apod.);
  - *Task Description Editor* - popis atributů úkolů a grafické vytváření souhrnných úkolů;
  - *Organisation Editor* - definování organizačních vztahů mezi agenty a přesvědčení agenta o schopnostech jiných agentů;
  - *Coordination Editor* - výběr sady protokolů koordinace, jimiž má být agent vybaven.
- *Visualization Tools* - vizualizační nástroje shromažďují informace o činnosti agenta i celého systému, umožňují zobrazit různé aspekty v reálném čase. Tyto nástroje usnadňují analýzu a debugovaní distribuovaného multiagentního systému.

Na základě specifikace provedené pomocí grafických nástrojů je automaticky vygenerován Java kód agentů a spouštěcí skripty. Specifické funkce agentů si mohou vývojáři sami doprogramovat v jazyce Java.

Nástroj ZEUS podporuje komunikaci mezi agenty pomocí starší verze FIPA ACL (Agent Communication Language). Je vhodný pro programovaní aplikací zaměřených na zřizování služeb, správu zdrojů/procesů, podporu managementu. Jedná se o open source projekt, takže ho lze využít zdarma. Na internetu je ke stažení kompletní instalační balík, dále manuál a tutoriál, které mohou usnadnit tvorbu prvních aplikací. Nevýhodou tohoto nástroje je, že nepodporuje mobilitu agentů a že již není dále rozvíjen. Poslední verze byla uvolněna v lednu 2006.

3.2 JADE

JADE (**J**ava **A**gent **DE**velopment Framework) [26] je softwarový framework plně implementovaný v jazyce Java. Zjednodušuje implementaci distribuovaných multiagentních systémů prostřednictvím middle-ware, který splňuje specifikaci FIPA, a sadou grafických nástrojů, které podporují fázi ladění i fázi nasazení projektu. Agentní platformu JADE lze rozdělit mezi více hostitelů s odlišnými operačními systémy. Konfigurace se provádí vzdáleně prostřednictvím grafického uživatelského rozhraní (GUI). Podle potřeby je možné agenty přesouvat nebo klonovat z jednoho hostitele na druhého. Jediný systémový požadavek je JRE (Java Runtime Environment) ve verzi 1.4.

Hlavním cílem platformy je podpora interoperability důsledným dodržováním FIPA specifikace týkající se samotné architektury platformy, stejně jako komunikační infrastruktury. JADE je navíc velmi flexibilní a aplikace mohou být přizpůsobeny pro použití na zařízeních s omezenými zdroji jako jsou PDA nebo mobilní telefony [2]. Komunikační model sestává z protokolů, obálek, ACL, jazyků, schémat kódování a ontologie. Také transportní protokoly jsou jasně vymezeny a plně integrovány. Agenti JADE používají speciální model spouštění založený na preemptivním plánování dynamicky spustitelných zásuvných modulů Javy, nazvaných chování. Tento model umožňuje programátorovi relativně snadno vyvíjet softwarové agenty, které jsou schopny flexibilního, reaktivního nebo proaktivního chování [1].

Platforma JADE nabízí vynikající škálovatelnost, přičemž se nezapomíná ani na podporu bezpečnosti a podporu webových služeb. Jeho výhodou je obliba mezi velkou skupinou programátorů, což se odráží i na množství dostupných technických manuálů a tutoriálů. Používá se od výukových programů v rámci univerzit po průmyslové nasazení (mobilní aplikace, internetové aplikace, podnikové aplikace, aplikace pro simulaci, síťový management, apod.). JADE je open source software distribuovaný pod licencí LPGL. Pro stažení kompletního instalačního balíku je potřeba se zaregistrovat na stránkách výrobce. Poslední verze byla uvolněna v březnu 2013.

3.3 AgentTool III

AgentTool III (zkráceně aT3) je grafické vývojové prostředí. Podporuje metodiku organizačně založeného multiagentního systémového inženýrství O-MaSE (Organization-based Multiagent Systems Engineering). Je nástupcem původního agentTool, který byl vyvinut v letech 2000 - 2001 v Air Force Institute of Technology. V současné době je rozvíjen jako projekt Multiagent & Cooperative Robotics (MACR) Laboratory na Kansaské státní univerzitě. Jedná se o plugin založený na jazyce Java a určený pro platformu Eclipse.

Cílem aT3 je podporovat a prosazovat O-MaSE, který umožňuje uživatelům vytvářet své vlastní přizpůsobené vývojové procesy. O-MaSE má tři struktury: metamodel, sadu metod fragmentů (set of methods fragments) a sadu pokynů, které umožňují uživatelům používat metody inženýrských technik k vytváření vlastních procesů. O-MaSE metamodel definuje klíčové pojmy potřebné k návrhu a implementaci multiagentních systémů. O-MaSE metody fragmenů zahrnují skutečné úkoly, které jsou vykonávány, a produkty, které jsou vyráběny. Pokyny O-MaSE definují, jaké jsou vzájemné vazby mezi metodami fragmentů, a tedy jak lze vytvářet vlastní procesy [4].

aT3 má čtyři komponenty, které jsou integrovány do jediného nástroje. Jsou to komponenty:

- *Graphical Editor* - definování modelu (Goal Model, Organization Model, Role Model, Domain Model, Agent Class, Protocol Model, Capability Model, Agent Plan Model, Policy Model);
- *Process Editor* - vytváření vlastních procesů založených na agentech;
- *Verification Framework* - zajišťuje konzistenci O-MaSE modelu na základě sady předdefinovaných pravidel;
- *Code Generation Facility* - automatické generování zdrojového kódu. Generátor je schopný generovat kód funkčního systému založeného na platformě JADE.

Modely se definují graficky pomocí diagramů, které jsou tvořeny objekty a vazbami mezi těmito objekty. Objekty se vybírají z palet nástrojů, které jsou odlišné pro každý typ modelu. U vložených objektů lze editovat jejich vlastnosti, metody, parametry, atd. Nabídka nastavení jednotlivých objektů se opět liší pro každý typ objektu. Na základě definic modelu je automaticky vygenerován Java kód. AgentTool III je navržen tak, aby ho bylo možné rozšířit o nové funkce přidáním vlastních rozšíření napsaných v jazyce Java.

Na Kansaské univerzitě je metodika O-MaSE použita v několika projektech včetně adaptivního bitevního informačního systému, projektu kooperativních robotických týmů nebo systému pro sledování a řízení rozsáhlých informačních systémů. Na webových stránkách AgenTool [23] lze nalézt aktuální informace o tomto nástroji, uživatelskou dokumentaci, pokyny k instalaci a řadu tutoriálů. Ke zprovoznění AgentTool III je potřeba mít nainstalovaný JRE 1.5.0 nebo vyšší a Eclipse alespoň ve verzi 3.4.2. Samotný plugin je ke stažení ze stránek výrobce. Poslední verze byla uvolněna v červnu 2011.

3.4 AgentBuilder

AgentBuilder je sada integrovaných nástrojů zaměřených na vytváření inteligentních softwarových agentů. AgentBuilder se skládá ze dvou hlavních komponent: Run-Time System a Toolkit. Run-Time System zahrnuje engine potřebný pro spuštění softwarových agentů, zatímco AgentBuilder Toolkit poskytuje vývojářům grafické nástroje, které mu umožní (využitím nabídek z panelů nástrojů, pop-up menu, combo boxů) snadný a rychlý vývoj inteligentních agentů a agentově orientovaného softwaru. Tyto nástroje zahrnují:

- *Project Control Tools* - nástroje pro organizaci a řízení projektu (Project Manager, Projet Dictionary Tool, Project Repository Manager);
- *Ontology Manager* - poskytuje nástroje pro analýzu problémů v rámci domény, vytvoření ontologie, konceptů a modelů objektů;
- *Agency Manager* - vytváření agentur, které jsou tvořeny dvěma nebo více agenty. Agenti spolu komunikují a vzájemně spolupracují při plnění úkolů. Agenti mohou být identičtí nebo specializovaní na plnění odlišných úkolů. Agency Manager umožňuje identifikovat a nastavovat typy agentů. Dále nabízí možnost run-time sledování činnosti agentů v systému. Vývojář tak může sledovat vzájemnou komunikaci agentů, kontrolovat agenty nebo spustit debugger;
- *Agent Manager* - poskytuje nástroje pro definování počátečního mentálního modelu agenta a jeho chování. Umožňuje nastavit přesvědčení, závazky, úmysly, možnosti a pravidla chování. Dále nabízí nástroje pro plánování a určení schopností agenta učit se;
- *Planning and Learning* - nástroj poskytuje možnost vytvořit specifické doménové moduly pro plánování a učení;
- *Agent Debugger* - zobrazuje v grafické podobě mentální model agenta, jeho vstupní a výstupní zprávy. Vývojář má k dispozici breakpointy a možnost krokovat, což mu usnadní odladit jednotlivé činnosti agenta.

AgentBuilder je komerční software nabízený ve dvou variantách - v Pro a Lite verzi. Pro verze má oproti Lite verzi vylepšenou podporu pro vytváření multiagentních systémů. Cena jedné licence je v případě Lite verze 95 $, resp. 895 $ za Pro verzi. Výrobce nabízí za zvýhodněnou cenu multilicenci pro dvacet současně pracujících uživatelů a výraznou slevu při nákupu softwaru pro akademické účely. Aktuální ceny jsou uvedeny na stránkách výrobce, kde lze nalézt ke stažení i zkušební verzi softwaru [20]. Instalační balík obsahuje vše potřebné pro spuštění včetně JRE. Tento nástroj umožňuje vývoj rozmanitých agentů. Agenti se používají například pro prohledávání internetu, pro nákup a prodej v aukcích, nebo je lze využít

pro složité distribuované řídící a monitorovací systémy. Poslední verze byla uvolněna v listopadu 2004.

3.5 Cougaar

Cougaar (Cognitive Agent Architecture) je open source architektura založená na Javě. Je výsledkem dlouholetého výzkumného programu agentury DARPA (Defense Advanced Research Projects Agency). Tato architektura poskytuje vývojářům framework pro implementaci rozsáhlých distribuovaných agentově orientovaných aplikací. Architektura byla navržena s hlavním důrazem na škálovatelnost a výkon. Navíc rozšiřuje agentní technologie zavedením kognitivního modelu, který napodobuje lidskou schopnost dynamického plánování řešení problému. Agent pro vyřešení daného problému využije své vlastní znalosti čerpané z předchozích zkušeností, relevantní data a podnikové procesy, poté určí nejlepší rozklad na konkrétní úkoly nezbytné pro splnění daného cíle [3]. Cougaar používá flexibilní komponentový model. Agenti jsou složeni z pluginů, které lze v případě potřeby dynamicky načítat nebo odpojovat. Pluginy mezi sebou komunikují pomocí sdíleného, distribuovaného datového prostoru nazývaného tabule (blackboard). Komunikace mezi agenty probíhá výměnnou zpráv. Bezpečnost je možné zajistit šifrováním, autentifikací nebo pomocí Java Security Manager.

Cougaar je velmi silná a robustní platforma, na níž lze vybudovat škálovatelný komplexní systém pro plánování a plnění úkolů v dynamických prostředích. Její robustnost byla prokázána vyvinutím distribuované vojenské plánovací logistické aplikace UltraLog. Ta se skládá z více než 1000 agentů distribuovaných na 100 hostitelích [5]. Cougaar je velmi složitá architektura, která sice nabízí obrovské možnosti, ale také vyžaduje značnou počáteční přípravu před zahájením programování. Pro psaní kódu aplikací se využívá IDE programu Eclipse. Instalační balík Cougaar lze bezplatně stáhnout z webových stránek [24]. Ke Cougaar je k dispozici mnoho dokumentace, včetně popisu architektury, programátorského průvodce a základních tutoriálů. Budoucí rozvoj této architektury bude zaměřený na webové služby a přidání alternativních protokolů zpráv. Poslední verze byla uvolněna v červenci 2012.

3.6 RETSINA

RETSINA (The **R**eusable **E**nvironment for **T**ask **S**tructured **I**ntelligent **N**etwork **A**gents) [30] je infrastruktura multiagentního systému vyvinutá v Laboratořích Softwarových Agentů v Institutu Robotiky na univerzitě Carnegie Mellon. RETSINA podporuje komunity heterogenních agentů. Vychází z předpokladu, že agenti mohou v systému vytvářet skupiny, které spolu vstupují do peer-to-peer interakcí.

Koordinace struktur v komunitě agentů vychází více ze vztahů mezi agenty, než jako důsledek vynucování samotné infrastruktury. V souladu s tímto předpokladem RETSINA nevytváří centralizované řízení v rámci MAS (Multi-Agent System). Místo přímého řízení agentů raději implementuje distribuované služby infrastruktury, které usnadňují interakce mezi agenty. Agenti spolu komunikují jazykem KQML (Knowledge Query and Manipulation Language). Funkční RETSINA architektura se skládá ze čtyř základních typů agentů:

- *Interface agent* - interakce s uživatelem, přijímá vstup od uživatele, zobrazuje uživateli výsledky;
- *Task agent* - pomáhá uživateli provádět úkoly, formuluje plány na řešení problémů a realizuje tyto plány pomocí koordinace a výměnou informací mezi ostatními agenty;
- *Information agent* - poskytuje inteligentní přístup k heterogenní množině informačních zdrojů;
- *Middle agent* - slouží jako zprostředkovatel mezi agenty, kteří požadují služby a agenty, kteří tyto služby nabízí.

Každého RETSINA agenta tvoří čtyři znovupoužitelné moduly:

- *Communication and Coordination module* - přijímá a interpretuje zprávy a požadavky od jiných agentů;
- *Planning module* - ze vstupní množiny cílů vytváří plán vedoucí ke splnění těchto cílů;
- *Scheduling module* - využívá strukturu úkolů vytvořenou modulem plánování a plánuje jejich vykonávání;
- *Execution module* - monitoruje vykonávání úkolů a zajišťuje, že akce jsou prováděny v souladu s výpočetními a jinými omezeními.

Implementační nástroj určený pro vývoj agentů a MAS se nazývá RETSINA AFC (Agent Foundation Classes). Knihovny jsou k dispozici pro jazyk C a jazyk C++. První z nich se hodí pro vývoj menších agentů, kteří jsou optimalizováni pro zařízení s limitovanými zdroji jako jsou mobilní telefony, PDA, senzory, apod. Knihovny založené na jazyku C++ se používají k vývoji agentů jakékoli složitosti. O této architektuře existuje mnoho dokumentů, článků a funkčních specifikací, včetně AFC Developers' Guide s mnoha praktickými příklady. Nástroj AFC je možné používat zdarma k akademickým a nekomerčním účelům. K jeho získání je nutné podepsat souhlas s licenčními podmínkami a zaslat jej na adresu Institutu Robotiky [21].

Architektura RETSINA byla aplikována v mnoha oborech týkajících se správy finančního portfolia, vyhledávání informací na webu, elektronického obchodu, bezdrátových a mobilních aplikací, plánování logistiky ve vojenských operacích, atd. Poslední verze byla uvolněna v listopadu 2002.

3.7 MadKit

MadKit (**M**ulti-**A**gent **D**evelopment **Kit**) je open source modulární a škálovatelná multiagentní platforma napsaná v Javě, která [27]:

- umožňuje vytvářet agenty a spravovat jejich životní cyklus;
- obsahuje organizační infrastrukturu vzájemné komunikace agentů;
- se vyznačuje vysokou mírou heterogenity v agentní architektuře. Neobsahuje předdefinovaný model agenta;
- obsahuje nástroje pro vytváření multiagentně založených simulací a distribuovaně založených aplikací.

Platforma je postavena na organizačním modelu AGR (Agent Group Role). MadKit agenti hrají takzvané role. Role je chápána jako abstraktní reprezentace funkce agenta, služby nebo označení v rámci skupiny. MadKit používá agenty k distribuovanému zasílání zpráv, řízení migrace, dynamické bezpečnosti a další správě systému. Tyto různé služby jsou v platformě reprezentovány jako role specifických skupin definovaných v abstraktní skupině. Služby agentů lze bez omezení nahradit za jiné, což zajišťuje vysokou úroveň škálovatelnosti. MadKit komunikace probíhá pomocí zasílání asynchronních zpráv a je založena na peer-to-peer mechanismech. Agenti v MadKit mohou být naprogramováni v jazyce Java, Scheme (Kawa), Jess nebo BeanShell. Použití JNI (Java Native Interface) by mělo umožnit vývoj agentů v jazycích C a C++.

MadKit je nabízen zdarma pod licencí GPL/LPGL. GPL licence se týká vývojářských nástrojů, LGPL licence se vztahuje na jádro a komunikační nástroje. Je ke stažení jako kompletní balík obsahující všechny potřebné knihovny a nástroje pro vývoj a provozování agentů a multiagentních systémů. Na webu je dostupná i dokumentace, FAQ, tutoriály a fórum. MadKit byl použit v mnoha projektech, které zahrnují širokou škálu aplikací. Jako příklad lze uvést simulaci hybridních architektur pro kontrolu podmořských robotů, hodnocení sociálních sítí nebo multiagentní řízení výrobní linky [1]. Poslední verze byla uvolněna v březnu 2013.

3.8 MASON

MASON (**M**ulti-**A**gent **S**imulator **O**f **N**eighborhoods (nebo Networks)) je jádro pro jednoprocesní diskrétní událostní simulace a soubor nástrojů pro vizualizaci. MASON je implementován v jazyce Java a je navržen tak, aby byl flexibilní pro širokou škálu jednoduchých simulací, přičemž hlavní důraz je kladen na „swarmové“ simulace, resp. na simulace velkého množství agentů. Mason obsahuje knihovny a volitelné sady nástrojů pro vizualizaci ve 2D a 3D, vytváření screenshotů nebo videí

(použitím Java Media Framework). MASON nabízí propracovanou grafickou konzoli (Console), grafické rozhraní, které usnadňuje uživateli ovládání běhu simulace.

MASON je výsledkem společného úsilí mezi evoluční výpočetní laboratoří a centrem pro sociální složitosti na Univerzitě George Masona (George Mason University's Evolutionary Computation Laboratory a the GMU Center for Social Complexity). Návrh systému MASON byl podřízen těmto cílům:

- malé, rychlé, snadno srozumitelné a jednoduše modifikovatelné jádro;
- samostatná, rozšiřitelná vizualizace ve 2D a 3D;
- modely jsou zcela nezávislé na vizualizacích, které mohou být přidány, odstraněny nebo kdykoli změněny;
- produkování stejných výsledků nezávislých na konkrétní platformě;
- možnost uložit na disk checkpoint (kontrolní bod) stavu modelu, který lze přenést na jinou platformu, kde lze obnovit běh modelu s vizualizací nebo bez vizualizace;
- podpora pro efektivní běh až milionů agentů bez vizualizace;
- podpora pro efektivní běh velkého množství agentů s vizualizací (omezeno pamětí počítače).

Naplnění těchto cílů umožňuje, aby byl MASON použit k různorodým výzkumným účelům od robotiky a strojového učení k multiagentním modelům sociálních systémů (politické vědy, historický vývoj, využití půdy, ekonomika, simulace městské dopravy, apod.) [8]. Na webových stránkách MASON [28] je možné stáhnout kompletní balík knihoven, instalační soubor Java3D Frameworku pro zobrazování ve 3D, rozsáhlou dokumentaci i tutoriál. MASON je open source software distribuovaný pod licencí Academic Free License. Poslední verze byla uvolněna v květnu 2013.

3.9 Repast Simphony 2.0

Repast Simphony 2.0 (zkráceně Repast S) je těsně integrovaný, interaktivní, multiplatformní modelovací systém založený na jazyce Java. Podporuje vývoj extrémně flexibilních modelů spolupracujících agentů určených pro běh na pracovních stanicích nebo v malých výpočetních clusterech. Existuje i verze Repast HPC (Repast for High Performance Computing) zaměřená na použití pro rozsáhlé distribuované výpočetní platformy a paralelní operace. Modely Repast Simphony mohou být vyvíjeny v několika různých formách včetně ReLogo (dialekt jazyka Logo), dále „point-and-click“ vývojových diagramů (flowcharts), Groovy nebo v Javě. Všechny tyto přístupy jsou úzce propojeny vývojovým prostředím Eclipse IDE, který je dodáván již předinstalovaný a přednastavený v instalačním balíku Repast Simphony 2. Balík obsahuje vše potřebné pro vývoj aplikací.

Nástroj Repast S byl vyvinut v Argonské národní laboratoři (Argonne National Laboratory) a je nabízen zdarma jako open source aplikace [29]. Umožňuje vizuálně specifikovat logickou strukturu modelů, prostorovou strukturu modelů, druhy agentů a jejich chování. Agenty Repast S je možné měnit za běhu [11]. Dále je možné vytvořit checkpoint stavu modelu a uložit ho do různých formátů včetně XML [19]. Běhové prostředí poskytuje [12]:

- GUI aplikaci pro vizuální nastavení prostředí a provádění operací s modelem;
- integrované 2D, 3D a mnoho dalších vizualizačních pohledů, které mohou zobrazovat aktuální stavové informace modelu;
- automatické připojení na zdroje dat různého typu (XML, CSV);
- automatické propojení s externími programy třetích stran pro statistické zpracování, analýzu a vizualizaci výsledků modelu;
- distribuování modelu i s běhovým prostředím jako spustitelné aplikace, pro další osoby, které mohou s modelem dále experimentovat.

Repast S byl úspěšně použit v mnoha aplikačních oblastech, jako například sociální věda, podpora prodeje produktů, zásobovací řetězce, simulace starověké pěší dopravy a mnoho dalších. K tomuto nástroji je možné nalézt dokumentaci včetně základních tutoriálů a FAQ, které usnadní začátky práce s nástrojem Repast S. Výhodou může být také fakt, že se na internetu nachází ke stažení mnoho hotových modelů. Navíc systém umožňuje importovat modely NetLogo. Poslední verze byla uvolněna v květnu 2012.

3.10 SeSAm

SeSAm (**S**hell for **S**imulated **A**gent Syste**m**s) je generické prostředí pro vývoj a simulaci multiagentních modelů založený na jazyce Java. Původně bylo vyvinuto na univerzitě ve Würzburgu. Od roku 2008 je nadále rozvíjeno na univerzitě v Örebro. Cílem vývojářů bylo poskytnout uživatelům bez hlubší znalosti programování nástroj pro snadné vytváření komplexních modelů. Tyto modely mohou obsahovat dynamické závislosti a emergentní chování. SeSAm poskytuje:

- snadné vizuální modelování;
- integrovanou grafickou simulační analýzu;
- flexibilní prostředí a definice situace;
- distribuované simulace v rámci lokální sítě;
- import vektorových a rastrových GIS souborů, import/export CSV souborů a textových souborů.

Hlavními entitami v modelu SeSAm jsou agenti, zdroje a svět. Jejich stav a chování se implementuje pomocí diagramů podobných UML. Jedná se tedy o vizuální

styl programování. Uživatel navrhuje simulaci z rozsáhlého počtu primitivních stavebních bloků. Uživatelské funkce, vlastnosti a vlastní datové typy umožňují škálování komplexních multiagentních simulací. Simulace různých situací se mohou slučovat do tzv. experimentů. SeSAm používá vlastní jazyk SeSAm-Impl a SeSAm-UML. Před samotným spuštěním simulace je model zkompilován, což vede k optimalizaci rychlosti běhu simulací. Samozřejmostí jsou nástroje pro shromažďování a vizualizaci dat. SeSAm lze doplnit množstvím pluginů, které umožňují přidat primitivy, datové typy, editory nebo FIPA ACL plugin umožňující komunikaci mezi simulací a skutečnými agenty (JADE). Existují i pluginy pro různé prostorové reprezentace jako 2D, 3D nebo vektorová data GIS [13].

SeSAm je nabízen ke stažení zdarma [31]. Jedná se open source aplikaci pod licencí LGPL. K projektu byla zpracována encyklopedie SeSAm-Wiki [32] obsahující dokumentaci, popis funkcí, pluginů a tutoriály. Multiagentní simulace SeSAm byly využity v mnoha oblastech jako např.:

- logistika (koordinace, optimalizace skladování);
- výroba (optimalizace plánů pro různé požadavky);
- doprava (zamezení dopravím zácpám, řízení světelné signalizace);
- biologie (pochopení chování hmyzu);
- zdravotní péče (optimalizace procesů, redukce nákladů, apod.).

SeSAm je velmi užitečný nástroj pro MAS simulace zejména pro komplexní modely s flexibilním chováním agentů a vzájemnou interakcí. Poslední verze byla uvolněna v září 2012.

3.11 Swarm

Swarm je softwarový balík pro vytváření multiagentních simulací komplexních systémů. Původně byl vyvinut v Santa Fe institutu a od roku 1999 je jeho rozvoj pod kontrolou neziskové organizace Swarm Development Group. Cílem vytvoření této platformy bylo poskytnout výzkumným pracovníkům základní architekturu pro vytváření simulací, které lze využít pro studium globálního a adaptivního chování v komplexních systémech. Swarm poskytuje knihovny znovupoužitelných komponent pro stavbu modelů, nástroje pro analýzu, zobrazování a kontrolu experimentů ve vytvořených modelech. Knihovny Swarm jsou napsány v jazyce Objective-C. Existuje však i verze Java Swarm, která umožňuje přistupovat ke knihovnám Objective-C pomocí jazyka Java. Modely lze tedy implementovat v obou těchto jazycích.

Základní jednotkou simulace je swarm. Jedná se o seskupení agentů, kteří provádějí určitou činnost. Taková jednotka je schopná stavět si svůj vlastní svět. Swarm je vlastně objekt, který disponuje určitou pamětí a plánovačem událostí. Objekty

Swarmu jsou hierarchicky organizovány do podoby podtříd (dílčích skupin, rojů). Každý Swarm je zodpovědný za řízení své paměti a zpracovávání zdrojů, které jeho objekty požadují pro svou činnost [16]. Swarm obsahuje tři typy objektů:

- *model* - tvorba a kontrola aktivit agenta v modelu;
- *pozorovatel* - sbírá informace od agentů;
- *agent* - agenti disponují akcemi a vlastnostmi.

Architektura Swarmu je tvořena jádrem, grafickým uživatelským prostředím a vytvořeným modelem. Tento koncept dovoluje, aby uživatel mohl přerušit běh simulace a individuálně prohlížet či měnit vnitřní stav agentů. V průběhu simulace je také možno přidávat nebo odebírat agenty.

Swarm je velmi silná a flexibilní platforma, na jejímž základě bylo vybudováno mnoho simulací v různých vědních oborech, jako např. ve fyzice, biologii, archeologii a ekonomii. Tyto možnosti s sebou nesou daň v podobě nároků na zkušenosti programátorů. Ti by měli mít znalosti jazyka Objective-C (případně Java) a měli by dobře znát metodiku objektového programování. Na internetu lze nalézt Swarm uživatelskou příručku, návody, Wiki, FAQ, demo aplikace a další dokumentaci [33] [18]. Swarm Development Group pravidelně pořádá konferenci Swarmfest, kde se scházejí vývojáři, uživatelé a výzkumní pracovníci, kteří společně diskutují o Swarm a dalších agentně založených platformách. Swarm je distribuován zdarma jako open source software s licencí GPL.

Tab. 3.1: Nástroje pro vytváření softwarových agentů

Platforma	Použití	Aktivní projekt	Licence	Programovací jazyk	Operační systém	Podpora uživatele	Ukázkové příklady
Zeus	Distribuované multiagentí simulace	Ne	Open source	Vizuální programování - generátor kódu	Windows; Linux; BSD; UNIX-like OSes; Solaris	Dokumentace; kontaktování vývojářů	Ne
JADE	Distribuované aplikace složené z autonomních entit	Ano	LGPL version 2	Java	Platforma Java	FAQ; zasílání novinek emailem; hlášení chyb; tutoriály; API; dokumentace	Ano
AgentTool III	Univerzální multiagentní systémy	Ano	N/A	Java	Platforma Java	Dokumentace; programátorský manuál; seznam publikací; pokyny k instalaci	Ne
AgentBuilder	Univerzální multiagentní systémy	Ano	Proprietární software; Zlevněné akademické licence	Java; C; C++	Windows; Linux; Sun Solaris; Platforma Java	Konzultace; trénink; FAQ; uživatelské manuály; hlášení chyb; zasílání novinek emailem	Ano
Cougaar	Komplexní multiagentí distribuované, škálovatelné, doménově nezávislé systémy	Ano	Cougaar Open Source License (COSL) - modifikovaná BSD licence	Java	Windows 98; Windows NT; Windows XP; Linux; Mac OS X; Java-1.4-kompatibilní PDA	FAQ; tutoriály; prezentace; dokumentace; vybrané reference; emailová podpora; veřejné forum; zasílání novinek emailem	Ano

RETSINA	Univerzální multiagentní systémy	Ano	CMU Licence; k nekomerčním a akademickým účelům zdarma	C/C++	Windows, SunOS, Linux	FAQ; dokumentace; příručka programátora; funkční specifikace	Ano
MadKit	Generická, flexibilní, škálovatelná platforma; Univerzální multiagentní platforma s podporou simulací založených na agentech	Ano	LGPL jádro; GPL vývojářské nástroje	Java, Scheme (Kawa), Jess, BeanShell, Python (jython), C/C++	Platforma JAVA	FAQ; dokumentace; online forum; hlášení chyb	Ano
MASON	Univerzální, fyzikální modelování, abstraktní modelování, AI/strojové učení	Ano	Academic Free License (open source)	Java	Platforma Java	Zasílání novinek; dokumentace; tutoriály; rozšíření třetích stran; seznam referencí; API	Ano
Repast Simphony 2.0	Univerzální simulace	Ano	BSD	Java, Grooby, ReLogo, Vizuální programování	Platforma Java	FAQ; dokumentace; tutoriály; zasílání novinek; hlášení chyb; externí nástroje; seznam referencí	Ano
SeSAm	Univerzální multiagentní simulace; výzkum, výuka, teorie grafů	Ano	LGPL	Vizuální programování	Java 5.0 nebo novější; Windows; Linux; Mac OS X	Tutoriály; zasílání novinek; FAQ; wiki; kontaktování vývojářů	Ne
Swarm	Univerzální multiagentní systémy	Ano	GPL	Java; Objective-C	Windows; Linux; Mac OS X	Wiki; tutoriály; dokumentace; FAQ; vybrané publikace; zasílání novinek	Ano

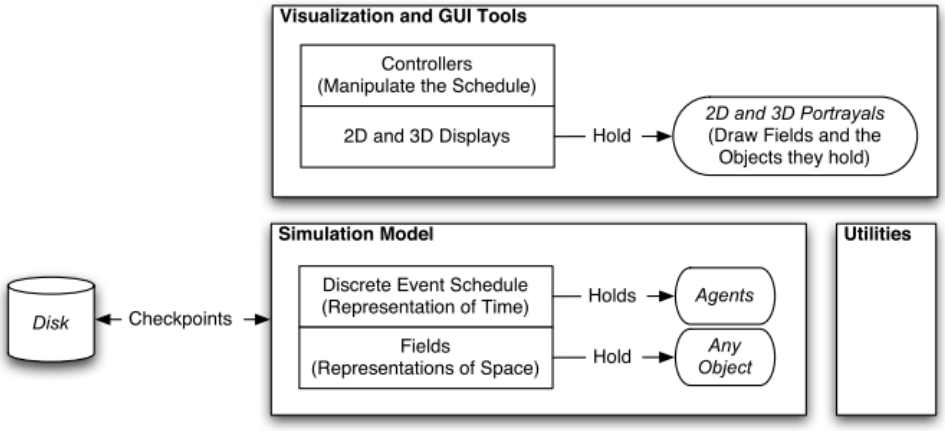
4 MASON

Stručné představení nástroje MASON bylo uvedeno na straně 22 v sekci 3.8. Následující řádky jsou zaměřeny na detailnější popis nástroje MASON. Jsou na nich uvedeny i informace o instalaci tohoto nástroje a spouštění simulací.

4.1 Popis prostředí

4.1.1 Architektura

MASON je založen na modulární vícevrstvé architektuře, jak lze vidět na obr. 4.1. Spodní vrstva je tvořena několika obecně použitelnými třídami, které poskytují: kvalitní generátor náhodných čísel, různé kolekce objektů, GUI widgety a nástroje pro vytváření snapshotů a videí. Dále následuje modelová vrstva, což je malá sada tříd zahrnujících diskrétní plánovač událostí, plánovací nástroje a mnoho typů polí (*fields*), do kterých se ukládají objekty včetně jejich umístění v prostoru. Vizualizační vrstva umožňuje grafickou vizualizaci a manipulaci s modelem. Mezi modelovou a vizualizační vrstvou je jasná dělící čára. Díky tomu je možné kdykoli oddělit model od vizualizace, uložit checkpoint na disk, přenést ho na jinou platformu a pokračovat v běhu simulace, třeba s úplně jinou podobou vizualizace.



Obr. 4.1: Základní prvky nástroje MASON: modelová a vizualizační vrstva [9].

4.1.2 Modelová vrstva

Jak už bylo zmíněno výše, modelová vrstva MASON není závislá na vizualizační vrstvě a může být od ní oddělena. Celý model je obsažen v jedné instanci uživatelsky definované podtřídy modelové třídy (*SimState*). Tato instance obsahuje diskrétní plánovač událostí, generátor náhodných čísel a žádné nebo více polí (*fields*).

- ***Agenti*** - MASON definuje agenta specifickým způsobem, a to jako výpočetní jednotku, která může být naplánována k vykonání určité akce a která může manipulovat s prostředím. Tato specifikace přímo neurčuje, že se agent musí v modelovaném prostředí fyzicky nacházet. Agent je „mozek“, který může mít hmotnou nebo nehmotnou podobu.
- ***Plánování*** - MASON neplánuje přímo události, které mají být vykonány agentem. Raději plánuje agenty samotné. Agenta je možné naplánovat k mnohonásobnému vykonávání různých funkcí pomocí anonymní obalové třídy (*wrapper class*). Plán může být rozdělen do více uspořádání, která lze dále dělit vzhledem k času (*timestep*), přičemž čas je modelován jako série diskrétních událostí. Agenti naplánovaní na daný čas v rámci dřívějšího uspořádání, budou mít přednost před agenty naplánovanými na stejný čas, ale v rámci pozdějšího uspořádání. MASON také poskytuje řadu obalů, které umožňují seskupit agenty dohromady, iterovat je, spouštět je paralelně v samostatných vláknech. Agenti mohou být plánováni ke spuštění ve vlastních vláknech asynchronních s plánovačem.
- ***Pole*** - propojují libovolné objekty nebo hodnoty s umístěním v nějakém pomyslném prostoru. Některá z těchto polí jsou jen něco jako obaly pro jednoduchá 2D nebo 3D pole. Jiná nabízejí volné vazby. Objekt může existovat ve více polích najednou. Stejný objekt se může objevit v některých polích i vícekrát. Použití polí je zcela dobrovolné a uživatel může přidávat i vlastní pole. MASON poskytuje pole pro:
  - 2D a 3D pole objektů, celých nebo reálných čísel. Mohou být ohraničená nebo toroidní, s šestihranným, trojúhelníkovým nebo čtvercovým uspořádáním;
  - 2D a 3D sítě, které jsou řídce obsazeny objekty. Mohou být ohraničené, neohraničené nebo toroidní, s šestihranným, trojúhelníkovým nebo čtvercovým uspořádáním;
  - 2D a 3D řídce obsazený spojitý prostor (souřadnice reálných čísel), který může být ohraničený, neohraničený nebo toroidní;
  - sítě (grafy), jejichž hrany mohou být orientované nebo neorientované, volitelně mohou být i ohodnocené nebo označené.

4.1.3 Vizualizační vrstva

Objekty ve vizualizační vrstvě mohou zobrazovat stav objektů v modelové vrstvě. K tomu je zapotřebí obalit modelovou třídu *SimState* pomocí třídy *GUIState*. Třída *GUIState* je tedy zodpovědná za připojení/odpojení *SimState* a za vytváření nebo nahrávání checkpointů z disku. Objekty ve vizualizační vrstvě jsou také plánovány

(např. okno je překresleno po změně modelu). O plánování se stará vlastní miniplánovač obsažený v *GUIState*, který je synchronizovaný s plánovačem modelu. To umožňuje, aby byla vizualizační vrstva naprosto oddělená od modelové vrstvy.

MASON provádí vizualizaci prostřednictvím jednoho nebo více displejů v podobě GUI oken, které poskytují 2D nebo 3D pohled na zobrazovaná pole. Každý displej obsahuje jedno nebo více zobrazení (*field portrayals*). Tato zobrazení jsou zodpovědná za vykreslování polí na obrazovku a dovolují uživateli zavolat inspektory (*inspectors*), což je GUI panel umožňující prozkoumat nebo modifikovat parametry objektů uložených v rámci polí.

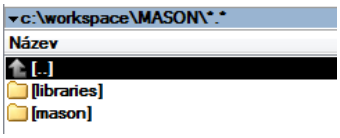
MASON poskytuje propracovanou konzoli (*Console*). Jedná se o grafické rozhraní, které uživateli usnadňuje ovládání simulace. Umožňuje spustit, zastavit, provést pauzu nebo krokovat běh simulace. Dále nabízí možnost schovat nebo vyvolat displeje, zobrazit inspektory, nahrát další simulace (každá má své vlastní *SimState*, *GUIState* a *Console*), nastavit zpoždění mezi jednotlivými kroky simulace, nastavit automatickou pauzu, atd. Z menu konzole lze kdykoliv načíst nebo uložit checkpoint, do něhož se ukládá aktuální stav všech agentů i celého prostředí.

4.2 Instalační soubory

- ***Balík MASON*** - obsahuje v sobě i ukázkové příklady a základní tutoriály.
  URL pro stažení: http://cs.gmu.edu/~eclab/projects/mason/mason.zip
  Aktuální verze: Version 17 - uvolněna v květnu 2013
- ***Volitelné knihovny*** - knihovny se používají pro generování videí, grafů, tabulek a pro rekompilaci modelu MASON.
  URL: http://cs.gmu.edu/~eclab/projects/mason/libraries.zip
- ***Java Platform (JDK)*** - podporovaná verze Javy je 1.3 nebo vyšší.
  URL pro stažení: http://www.oracle.com/technetwork/java/javase/downloads/index.html
  Aktuální verze: Java SE Development Kit 7u21
- ***Java 3D API*** - nutné pro zobrazování simulací ve 3D.
  URL pro stažení: https://java3d.java.net/binary-builds.html
  Aktuální verze: Java 3D 1.5.2
- ***Eclipse*** - MASON není nijak vázaný na vývojové prostředí Eclipse. Vývojář si může zvolit jiný vývojový nástroj, nebo ho vůbec nemusí používat.
  URL pro stažení: http://www.eclipse.org/downloads/
  Aktuální verze: Eclipse Classic 4.2.2

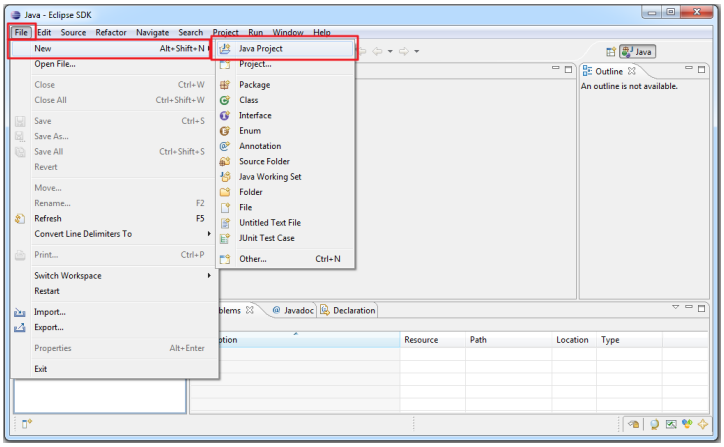
4.3 Vytvoření projektu a spuštění ukázkové simulace

1. V adresáři workspace programu Eclipse vytvořte podadresář MASON. V průzkumníku (nebo jiném správci souborů) do něj rozbalte balík *mason.zip* a volitelné knihovny *libraries.zip* (obr. 4.2).



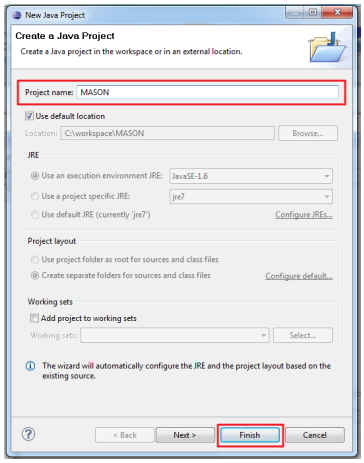
Obr. 4.2: Rozbalené adresáře ve workspace programu Eclipse.

2. Spusťte Eclipse a vyberte adresář workspace.
3. V Eclipse zvolte v nástrojové liště **File** -> **New** - **Java Project** (obr. 4.3).



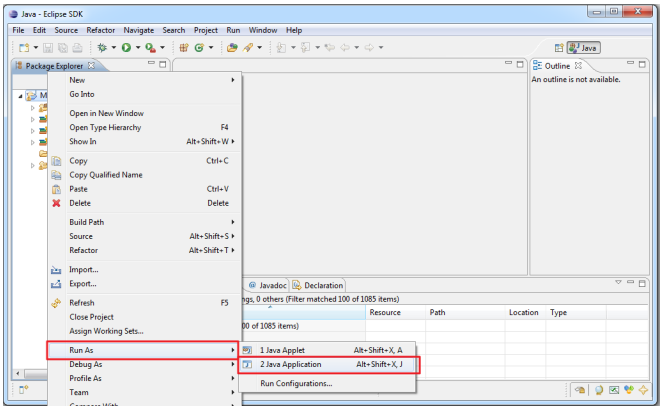
Obr. 4.3: Vytvoření nového projektu - krok 1.

4. Pojmenujte projekt MASON a klikněte na tlačítko **Finish** (obr. 4.4).



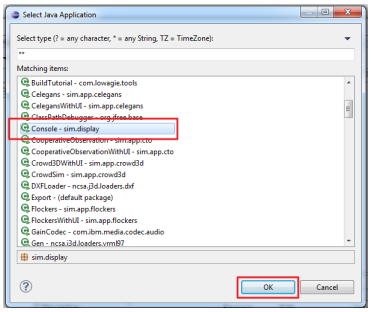
Obr. 4.4: Vytvoření nového projektu - krok 2.

5. V levé části obrazovky klikněte v **Package Exploreru** pravým tlačítkem myši na projekt MASON a zvolte na **Run As -> Java Aplication** (obr. 4.5).



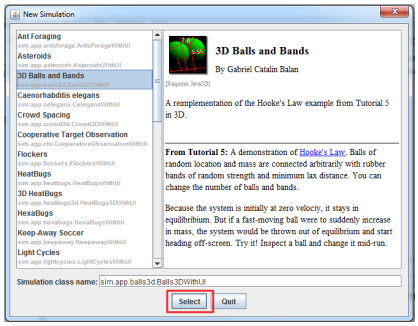
Obr. 4.5: Spuštění ukázkového příkladu - krok 1.

6. Eclipse vyhledá všechny spustitelné aplikace v rámci projektu MASON. Zvolte aplikaci **Console** a kliněte na tlačítko **OK** (obr. 4.6).



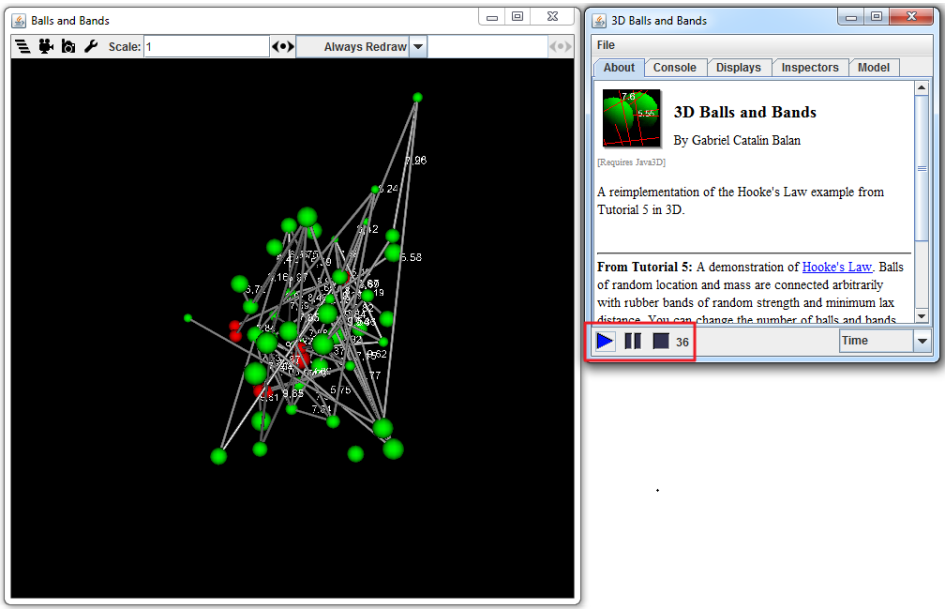
Obr. 4.6: Spuštění ukázkového příkladu - krok 2.

7. Spustí se nové okno MASON s nabídkou simulací ke spuštění. Vybranou simulaci spusťte kliknutím na tlačítko **Select**. (obr. 4.7).



Obr. 4.7: Spuštění ukázkového příkladu - krok 3.

8. Po spuštění simulace se objeví dvě samostatná okna. Jedno obsahuje ovládací prvky běhu simulace a druhé slouží pro vizualizaci. Běh modelu se spustí kliknutím na tlačítko **Play** (obr. 4.8).



Obr. 4.8: Spuštění ukázkového příkladu - krok 4.

5 REPAST SIMPHONY 2.0

Tato kapitola se zabývá detailním popisem nástroje Repast Simphony 2.0 včetně informací o instalaci. Doplňuje tak informace uvedené na straně 23 v sekci 3.9.

5.1 Popis prostředí

5.1.1 Architektura

Repast Simphpony 2.0 je založen na vysoce modulární vícevrstvé architektuře. Každý modul je nezávislý plugin, který může být jednoduše připojen nebo odpojen zapsáním několika řádků v XML. To umožňuje v případě potřeby nahradit jednotlivé moduly (např. moduly pro logování, sítě, časové plánování) za jiné.

Při návrhu architektury byl kladen důraz na striktní oddělení specifikace modelu, samotného běhu modelu, vizualizace a ukládání dat. Vznikl tak runtime engine pro vykonání modelu, runtime rozhraní pro vizualizaci modelu, *freeze dryers* pro ukládání checkpointů. Specifikace modelu se definuje ve vývojovém prostředí Eclipse a je podporováno několik jazyků.

5.1.2 Framework

Základem architektury je Simphony Application Framework (SAF), který je k dispozici zdarma jako open source. SAF jednodušuje běžné úkoly spojené s vývojem aplikací, jako je vytvoření a nastavení menu, panelů nástrojů, rozvržení okna aplikace, apod. Aplikace založené na SAF obvykle respektují návrhový vzor *model-view-controller* doplněný třídou, která zprostředkovává spojení mezi různými částmi aplikace. Implementace runtime rozhraní Repast Simphony je tedy tvořena třemi komponentami: uživatelským rozhraním, enginem simulační infrastruktury a aplikační třídy, která mezi nimi zprostředkovává interakci.

5.1.3 Jádro

Jádro Repast Simpony je plugin, který poskytuje významné funkce pro simulaci, jako jsou: plánovač času, správa prostoru, aktivace chování a generátor náhodných čísel. Diskrétní časový plánovač používá reálná čísla s dvojitou přesností. Události naplánované na stejný čas se spouští v pořadí na základě jejich nastavené priority. Repast plánovač nabízí imperativní i deklarativní plánování pro sekvenční nebo souběžné aktivity.

Správa prostoru využívá kontexty (*contexts*) a projekce (*projection*). Kontexty jsou hierarchicky vnořené a pojmenované kontejnery, které uchovávají modelové objekty. Modelovým objektem může být jakýkoli objekt Javy, včetně jiných kontextů. Typicky to ale bývá objekt agent. Komponenty modelu mohou být součástí mnoha kontextů. Hierarchické řazení znamená, že modelová komponenta obsažená v určitém kontextu, je také součástí všech kontextů rodiče tohoto kontextu. V případě zájmu může hierarchické řazení specifikovat sám návrhář modelu.

Kromě hierarchického ražení kontexty také podporují projekce. Projekce jsou pojmenované sady vztahů mezi prvky kontextu. Jako příklad projekce lze uvést projekční síť, která uchovává informace o vazbách mezi členy určitého kontextu. Členové tohoto kontextu se pak mohou ptát, s kým jsou propojeni, a kdo je na ně napojený.

Kontexty a projekce mohou využít tzv. *watchers* k aktivaci chování. Mechanismus *watcher* poskytuje návrhový vzor pro reaktivní chování agentů, které je výpočetně efektivnější než polling sady agentů ve stanovených intervalech [10]. Oba případy plánování umožňují souběžné i paralelní zpracování operací.

5.1.4 Eclipse

Repast Simphony využívá Eclipse jako primární vývojové prostředí. Eclipse používá podobnou architekturu založenou na pluginech jako Repast. Repast Simphony pluginy určené pro Eclipse poskytují nástroje, zobrazení (tj. okna) a perspektivy pro vytvoření řady specifických modelových komponent. Ty zahrnují obecné Repast projekty, ReLogo projekty, flowcharts (vývojové diagramy) a ReLogo agenty. Pluginy také umožňují spouštět modely, debuggovat a vytvářet samostatné instalační balíky určené pro nasazení.

5.1.5 Systém sběru dat

Systém sběru dat je určen ke shromažďování a ukládání informací o běžící simulaci. Systém od každého agenta shromažďuje předdefinované sady dat v každém časovém kroku. Nashromážděná data reprezentují stav simulace v určitém čase.

5.1.6 Programovací jazyky

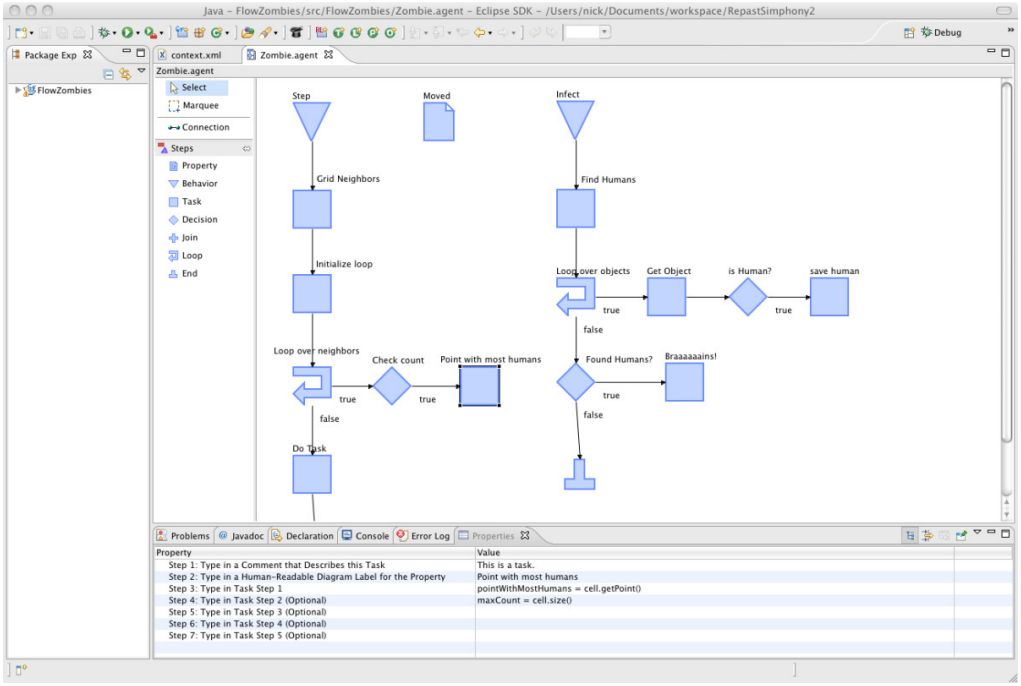
- ***Repast Java*** - dovoluje programátorovi vytvářet nejflexibilnější modely simulace. Agentem může být instance každé obyčejné třídy.

- ***Relogo*** - ReLogo obaluje velkou část funkcionality knihoven Repast Simphony do sémanticky jednoduchého ale efektivního balíku. Cílem je poskytnout návrhářům rychlou cestu, jak vyvíjet simulace, které mohou být později škálovány použitím funkcí obsažených v plné knihovně Repast Simphony.

  ReLogo obsahuje předpřipravené třídy, které pocházejí z původního programovacího jazyka Logo. Základními stavebními kameny jsou pohybliví agenti označovaní jako želvy (*turtles*), kteří se mohou pohybovat po dlaždicích (*patches*). Dále zde existují spoje (*links*), které umožňují modelovat vztahy mezi agenty, případně jejich skupiny. Poslední třídu tvoří pozorovatelé (*observers*), kteří umožňují celkové řízení projektu.

  ReLogo je založeno na jazyce Groovy a používá stejnou syntaxi. Implementuje však základní konstrukce a primitiva známé z jazyka NetLogo. Při psaní simulace v jazyce ReLogo může programátor kdykoli plynule přejít k psaní kódu v jazyce Java nebo Groovy a získat tak přístup k pokročilejším funkcím. Programátoři také mohou použít libovolnou Java nebo Groovy knihovnu bez použití speciální syntaxe. Repast nabízí i nástroj pro automatickou konverzi modelů NetLogo do modelů ReLogo.

- ***Repast Flowchart*** - umožňuje v grafickém prostředí vizuální tvorbu simulací pomocí skládání a propojování bloků. Na obr. 5.1 je vidět příklad vývojového diagramu, který může být použit k zakódování vlastností a chování agenta.



Obr. 5.1: Ukázka flowchart zobrazující chování Zombie agenta [10].

5.1.7 GIS

Geografické odkazování se poskytuje prostřednictvím geografických projekcí. Ty, stejně jako ostatní prostorové projekce v rámci Repast Simphony, souvisejí s pozicí agentů v prostoru. Reprezentace agenta v geografické projekci odpovídá specifickému geografickému typu, jako jsou body, čáry a polygony. Repast Simphony používá GeoTools (GeoTools – The Open Source Java GIS Toolkit).

5.1.8 2D a 3D vizualizace

Repast Simphony obsahuje moduly pro 2D a 3D vizualizaci. Ty umožňují interaktivní sledování běžících modelů ve 2D nebo 3D. Efektivní vizualizace je založena na dvoufázovém přístupu. V první fázi jsou všichni agenti aktualizováni, aby odráželi aktuální stav modelu. Ve druhé fázi je pak celá scéna vykreslena. K 2D vizualizacím se využívá OpenGL, resp. JOGL (Java Binding for the OpenGL API). Knihovna pro vizualizaci umožňuje uživateli snadno vytvářet různé geometrické tvary, importovat vektorovou grafiku, importovat obrázky a různě je stylovat. Také podporuje výběr objektů, zoomování a posouvání. Jednoho agenta je možné zobrazovat v různých topologiích (sítě, mřížky) a různých stylech. Ve 2D zobrazení může agent např. vypadat jako červené kolečko a ve 3D jako modrá kostka. Implementace 2D i 3D zobrazení používají struktury grafu scény. Jednotlivý agenti jsou ve scéně reprezentováni jako geometrické uzly. GIS zobrazení je podporováno v obou těchto implementacích.

5.1.9 Freeze drying

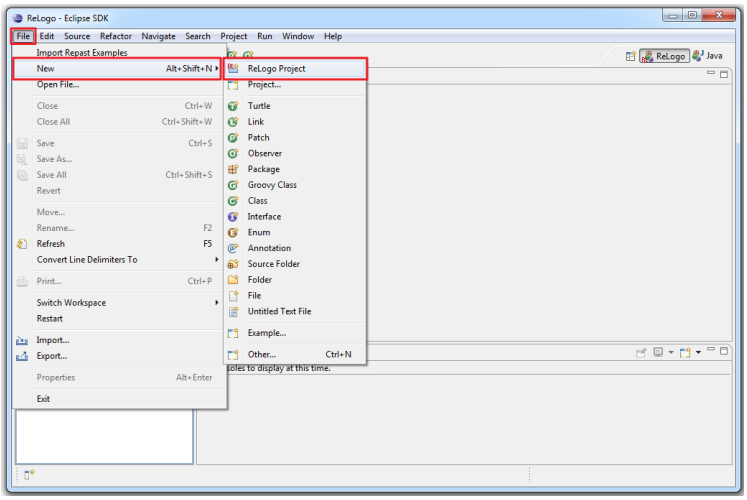
Repast Simphony umožňuje uživateli vytvořit checkpoint stavu simulace, který lze později znovu načíst a spustit. Je podporován tabulkový formát a XML formát. Data uložená v tabulce mají pro uživatele tu výhodu, že je možné je jednoduše nahrát do standardních databází, nebo je offline editovat načtením v běžných programech (např. Microsoft Excel). Pro ukládání dat do XML se využívá knihovna Xstream. Uložený checkpoint obsahuje aktuální stav všech agentů i celého prostředí.

5.2 Instalační soubory

- ***Repast Simphony 2.0*** - instalační soubor zahrnuje vývojové prostředí Eclipse a všechny potřebné knihovny pro vývoj ReLogo, Flowchart, Groovy nebo Java simulací. Obsahuje v sobě i ukázkové příklady a základní tutoriály. Velikost instalačního souboru je 367 MB.
  URL pro stažení: http://repast.sourceforge.net/download.html
  Aktuální verze: uvolněna v květnu 2012

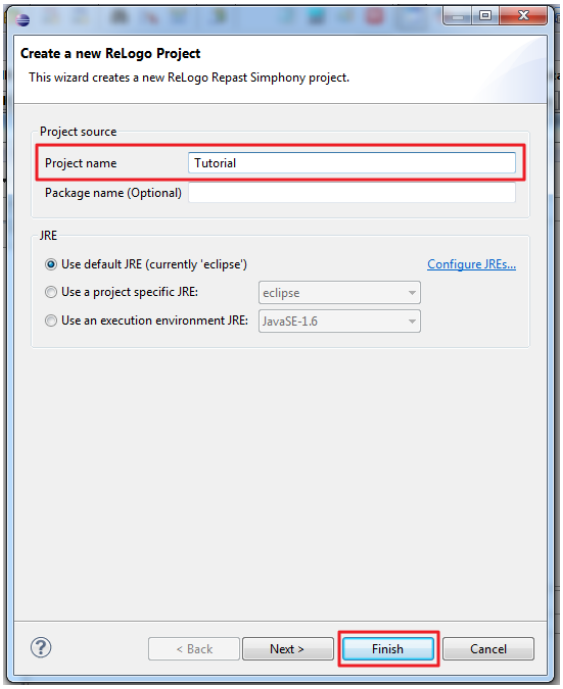
5.3 Vytvoření projektu a spuštění ukázkové simulace

1. Spusťte Eclipse a vyberte adresář workspace.
2. V Eclipse zvolte v nástrojové liště **File** -> **New** - **ReLogo Project** (obr. 5.2).



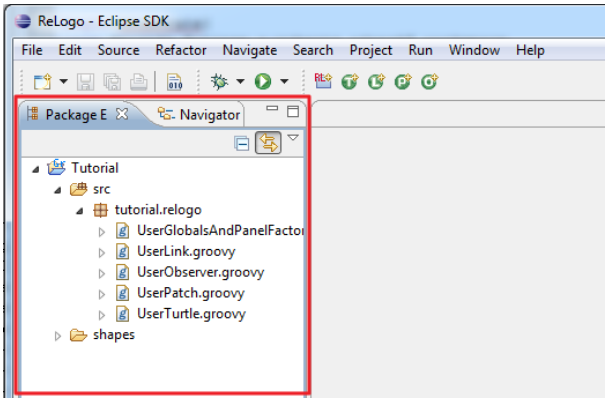
Obr. 5.2: Vytvoření nového projektu - krok 1.

3. Pojmenujte projekt např. Tutorial a klikněte na tlačítko **Finish** (obr. 5.3).



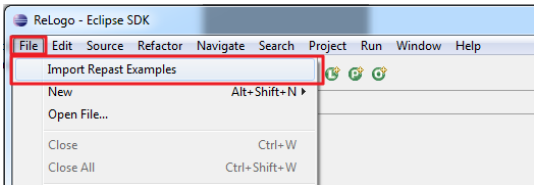
Obr. 5.3: Vytvoření nového projektu - krok 2.

4. V levé části obrazovky je vidět v **Package Exploreru** připravený prázdný projekt. Můžete začít vytvářet vlastní simulaci (obr. 5.4).



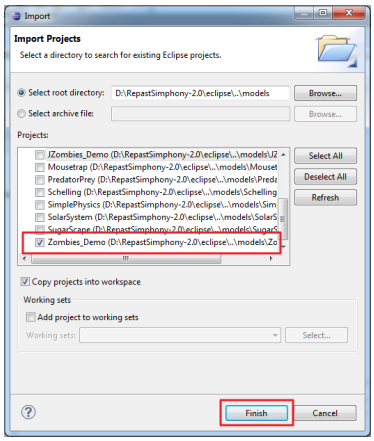
Obr. 5.4: Vytvoření nového projektu - krok 3.

5. Součástí standardní instalace Repast Simphony 2.0 jsou i ukázkové příklady. Pro naimportování ukázkového příkladu do Eclipse zvolte v nástrojové liště **File** -> **Import Repast Examples** (obr. 5.5).



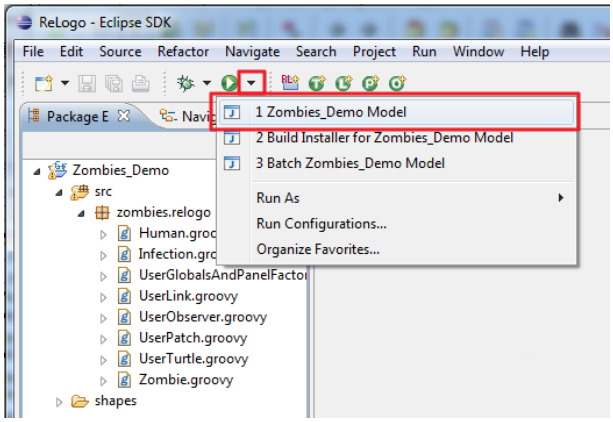
Obr. 5.5: Import ukázkového příkladu - krok 1.

6. Objeví se okno s nabídkou projektů s ukázkovými příklady. Eclipse automaticky nabízí cestu k projektům uloženým v instalačním adresáři s Repast Simphony 2.0. Vyberte požadovaný projekt a kliněte na tlačítko **Finish** (obr. 5.6).



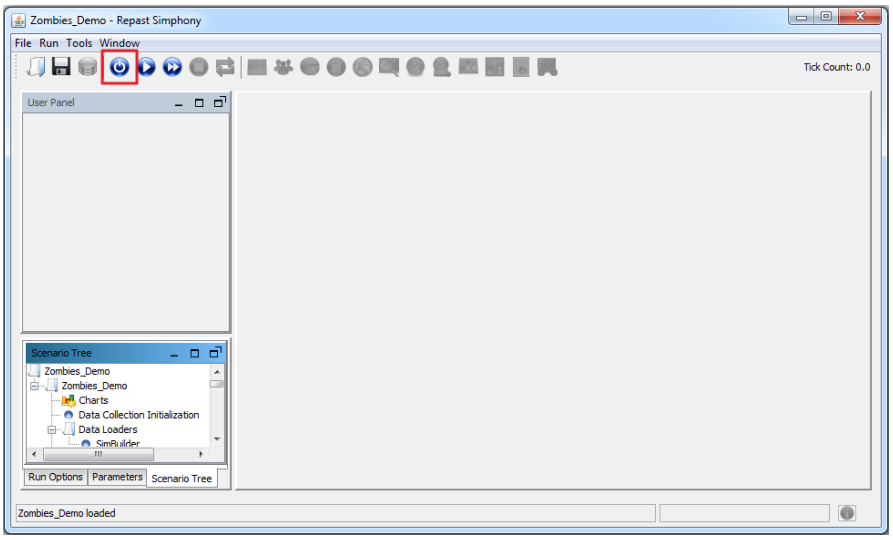
Obr. 5.6: Import ukázkového příkladu - krok 2.

7. V nástrojové liště klikněte na šipku směřující dolů a z rozbalovací nabídky spusťte model ukázkového příkladu (obr. 5.7).



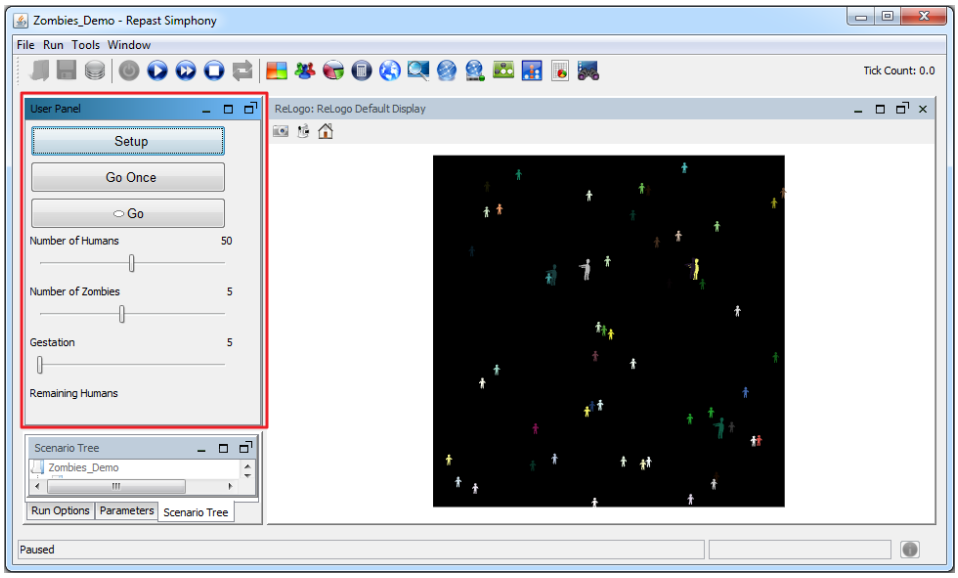
Obr. 5.7: Spuštění ukázkového příkladu - krok 1.

8. Po spuštění modelu se objeví nové okno s ovládacími prvky programu Repast Simphony 2.0. Model inicializujte tlačítkem **Initialize Run** (obr. 5.8).



Obr. 5.8: Spuštění ukázkového příkladu - krok 2.

9. Po inicializaci modelu se objeví uživatelem nadefinované ovládací prvky simulace. Pro nastavení modelu klikněte na tlačítko **Setup**. Běh modelu můžete ovládat tlačítky **Go Once** a **Go**. První jmenované tlačítko umožňuje běh simulace krokovat (obr. 5.9).



Obr. 5.9: Spuštění ukázkového příkladu - krok 3.

6 POROVNÁNÍ VYBRANÝCH NÁSTROJŮ

Porovnání nástrojů předcházelo vytvoření dvou simulací. První simulace, nazvaná Zombies, je modifikace simulace popsané v tutoriálu Relogo getting started guide [15]. Úloha má toto zadání: „V prostředí se nachází určitý počet lidí a zombie, kteří se v prostředí pohybují nahodile. Při střetu zombie s člověkem je člověk infikován a po určitém počtu kol se přemění v zombie.“ Simulaci jsem nejprve vytvořil v nástroji Repast Simphony 2.0 v jazyce ReLogo. Poté jsem ji namodeloval i v nástroji MASON v jazyce Java. Druhá simulace má toto zadání: „Agent se nahodile pohybuje ve 2D prostředí.“ Vytvoření této jednoduché simulace mělo posloužit k tomu, aby bylo možné porovnat, kolik je potřeba řádků zdrojového kódu k vytvoření modelu simulace s vizualizací.

Pojetí času

V obou porovnávaných nástrojích probíhá běh modelu krokově. V případě, že je agent v daném kroku naplánován, je vyvolána metoda definující akci/akce agenta.

Programovací jazyk

MASON i Repast Simphony 2.0 používají k vytváření modelů objektově orientovaný přístup. Základem obou nástrojů je jazyk Java, který je v případě Repastu S doplněn možností modelovat v jazyce ReLogo (napsaném v jazyce Groovy) nebo pomocí flowchart.

Zdrojový kód nutný k definici modelu a vizualizaci simulace

Na straně 54 v příloze B, resp. na straně 58 v příloze C, jsou ukázky zdrojových kódů jednoduchých simulací. Ty mají sloužit pro představu, kolik kódu je potřeba napsat, aby bylo možné spustit simulaci včetně vizualizace. Z těchto kódů je patrné, že k vytvoření stejné simulace v Repast S je potřeba zhruba třetina řádků zdrojového kódu oproti simulaci nadefinované v MASONu. Na tomto místě je vhodné upřesnit, že simulace v Repastu S byla vytvořena v jazyce ReLogo.

Komunikace mezi agenty

Ani jeden z nástrojů neobsahuje speciální API pro komunikaci mezi agenty. Jednotliví agenti mezi sebou komunikují voláním metod.

Definice prostředí

Oba nástroje umožňují definovat různorodá prostředí světa. Prostředí může být tvořeno mřížkou, sítí nebo jako kontinuální prostor ve 2D i 3D dimenzi. Jedná se o nepohyblivý prvek modelu, který se nemění s časem.

Rychlost spouštění simulace

Doba potřebná ke spuštění simulací je mnohem menší u nástroje MASON. Měřeno od okamžiku, kdy se v programu Eclipse klikne na tlačítko Run, do doby, kdy se objeví grafické okno s ovládacími prvky simulace. V případě nástroje MASON je tato doba cca jedna sekunda. U Repastu S trvá cca devět sekund, než se objeví grafické okno, přičemž je potřeba model simulace ještě inicializovat k tomu, aby se objevily ovládací prvky simulace. Inicializace zabere dalších cca sedm sekund. Tuto pomalost kompenzuje Repast S tím, že programátorovi dává možnost nadefinovat si vlastní ovládací prvky v potomkovi třídy *AbstractReLogoGlobalsAndPanelFactory*, které mu umožní měnit parametry (proměnné) modelu bez toho, aniž by grafické okno vypínal.

Ukládání stavu běžících simulací

Nástroje shodně nabízejí možnost uložit kdykoli na disk checkpoint aktuálního stavu všech agentů i celého prostředí. Ten lze pak opětovně načíst a obnovit běh modelu. Repast Simphony 2.0 navíc podporuje více formátů pro uložení dat.

Ovládání běhu simulace

Běh simulace lze v obou nástrojích ovládat pomocí grafického uživatelského rozhraní. Grafické okno MASON standardně obsahuje tlačítka pro spuštění, zastavení, krokování nebo stopnutí simulace. V případě Repast S je potřeba tato tlačítka nadefinovat ve zdrojovém kódu simulace. Oba nástroje také umožňují nastavit zpoždění mezi jednotlivými kroky simulace, automatickou pauzu nebo stopnutí běhu simulace, vytvořit screenshot, případně běh simulace uložit jako video soubor. V Repast S lze navíc před inicializací modelu simulace nastavit v grafickém okně rozměry prostředí (načítané z XML souboru).

Debugging

Ani jeden z těchto nástrojů neposkytuje speciální funkce pro debugging. U obou nástrojů může být užitečným pomocníkem při ladění simulace inspektor, zobrazující

aktuální stav agentů. Dále je programátor odkázán na standardní nástroje debuggingu obsažené v nástroji Eclipse.

7 ZÁVĚR

V této práci bylo představeno jedenáct nástrojů určených pro modelování softwarových agentů. Při výběru nástrojů pro srovnání byly upřednostněny nástroje, které dovolují vývojáři používat zdarma. Převážná část těchto nástrojů je implementována v jazyce Java. To má velkou výhodu v tom, že softwaroví agenti vytvoření v těchto nástrojích se stávají nezávislí na konkrétním operačním systému.

Při troše zjednodušení by šlo srovnávané nástroje rozdělit do tří kategorií. Jedna kategorie jsou nástroje určené pro vytváření multiagentních simulací. Ty najdou využití převážně u výzkumných pracovníků, kterým se vývojáři snaží co nejvíce usnadnit vytváření modelů simulací. Simulace je tak možné definovat vizuálně nebo pomocí speciálně navržených programovacích jazyků. Do druhé kategorie spadají nástroje, které poskytují vše potřebné pro vytváření a provozování distribuovaných systémů založených na agentech. Takto založené systémy jsou využívány pro vojenské účely i v komerční sféře. Třetí skupinu tvoří univerzální nástroje, pomocí nichž lze vytvářet jak simulace, tak agentně založený software pro jakékoli účely.

Ze srovnávaných nástrojů byly vybrány dva nástroje k detailnějšímu popisu a srovnání. Jedná se o nástroje MASON a Repast Simphony 2.0. Tyto nástroje svým zaměřením spadají do první kategorie. Jsou tedy určeny k vytváření simulací. Nástroje jsou si v mnoha oblastech velmi podobné. Nabízejí např. možnost rozšířit své funkce pomocí pluginů, vizualizovat běh modelu, vytvářet videa a screenshoty. Dále umožňují uložit na disk checkpoint (kontrolní bod) stavu modelu, který lze přenést na jinou platformu, a pokračovat v běhu modelu. Silnou stránkou obou nástrojů je fakt, že běh modelu simulace je zcela oddělen od vizualizační vrstvy. Model lze tak spustit bez vizualizace, s vizualizací nebo s více druhy vizualizací, mezi kterými lze přepínat.

Jedním z cílů práce bylo prakticky si vyzkoušet vytváření modelů simulací. To bylo splněno tím, že v každém z těchto nástrojů byly namodelovány dvě simulace se stejným zadáním. To ukázalo, že Repast Simphony 2.0 je při použití programovacího jazyka ReLogo mnohem úspornější, co se týká počtu řádků zdrojového kódu potřebných k vytvoření simulace s vizualizací. To ale neznamená, že by byl tento nástroj lepší. Pro plnohodnotné využití všech funkcí je potřebovat modelovat simulace v jazyce Java, stejně jako je tomu u nástroje MASON. Jazyk ReLogo ocení pro svou srozumitelnost a jednoduchost lidé bez zkušeností s programováním v Javě.

Součástí práce jsou i informace o instalaci nástrojů MASON a Repast Simphony 2.0 a základní manuály pro práci s těmito nástroji, resp. pokyny pro vytvoření projektu a spouštění ukázkových simulací. Případný zájemce o tuto problematiku nalezne v seznamu literatury odkazy na podrobnější informace.

SEZNAM ZKRATEK

ACL Agent Communication Language

AFC Agent Foundation Classes

AGR Agent Group Role

API Application Programming Interface

BSD Berkeley Software Distribution

CMU Carnegie Mellon University

CSV Comma Separated Values

DARPA Advanced Research Projects Agency

FAQ Frequently Asked Questions

FIPA The Foundation for Intelligent Physical Agents

GIS Geographic Information System

GPL General Public License

GUI Graphical User Interface

IDE Integrated Development Environment

JADE Java Agent DEvelopment Environment

JESS Java Expert System Shell

JNI Java Native Interface

JOGL Java Binding for the OpenGL API

JRE Java Runtime Environment

KQML Knowledge Query and Manipulation Language

LPGL Lesser General Public License

MAS Multi-Agent System

MIME Multipurpose Internet Mail Extensions

O-MaSE Organization-based Multiagent Systems Engineering

PDA Personal Digital Assistant

SAF Simphony Application Framework

TCL Tool Command Language

UML Unified Modeling Language

XML Extensible Markup Language

LITERATURA

[1] BĂDICĂ, Costin; BUDIMAC, Zoran; BURKHARD, Hans-Dieter; IVANOVIĆ, Mirjana. Software agents: Languages, tools, platforms. *Computer Science and Information Systems.* 2011, roč. 8, č. 2, s. 255-298. ISSN 1820-0214. DOI: 10.2298/CSIS110214013B.

[2] BORDINI, Rafael H.; BRAUBACH, Lars; DASTANI, Mehdi; SEGHROUCHNI, Amal El Fallah; GOMEZ-SANZ, Jorge J.; LEITE, Joao; O'HARE, Gregory; POKAHR, Alexander; RICCI, Alessandro. A Survey of Programming Languages and Platforms for Multi-Agent Systems (2006). DOI: 1874/134570.

[3] FANG, Frank; REED, Elaine; DICKASON, David K.; SIMIEN, H. J.; WULFF, Michelle L.: *Technology Review of Multi-Agent Systems and Tools.* Millington, TN : Navy Personnel Research, Studies, and Technology Department. 2005.

[4] GARCÍA-OJEDA, Juan C.; DELOACH, Scott A.; ROBBY: agentTool III: From Process Definition to Code Generation. *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '09).* 2009, vol. 2, Budapest, Hungary, s. 1393-1394. ISBN: 978-0-9817381-7-8

[5] HELSINGER, Aaron; THOME, Michael; WRIGHT, Todd: Cougaar: A Scalable, Distributed Multi-Agent Architecture. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics.* 2004, vol. 2, s. 1910-1917. DOI: 10.1109/ICSMC.2004.1399959

[6] VOLNÁ, Eva: *Úvod do problematiky multiagentních systémů.* První vydání. Ostrava: Ostravská univerzita v Ostravě, 2005. 51 s.

[7] RUSSELL, Stuart J.; NORVIG, Peter: *Artificial intelligence: a modern approach.* 2nd edition. New Jersey: Prentice Hall, 2003, xxviii, 1081 s. ISBN: 01-379-0395-2.

[8] LUKE, Sean; CIOFFI-REVILLA, Claudio; PANAIT, Liviu; SULLIVAN, Keith; BALAN, Gabriel: MASON: A Multi-Agent Simulation Environment. *SIMULATION.* 2005, vol. 81, no. 7, s. 517-527. ISSN: 00375497. DOI: 10.1177/0037549705058073

[9] LUKE, Sean; CIOFFI-REVILLA, Claudio; PANAIT, Liviu; SULLIVAN, Keith: *MASON: A new multi-agent simulation toolkit.* Virginia, George Mason University, Department of Computer Science and Center for Social Complexity. 2055.

[10] NORTH, M.J.; COLLIER, N.T.; OZIK, J.; TATARA, E.; ALTAWEEL, M.; MACAL, C.M.; BRAGEN, M.; SYDELKO, P.: Complex Adaptive Systems Modeling with Repast Simphony *Complex Adaptive Systems Modeling.* 2013, Springer, Heidelberg, FRG

[11] MACAL, Charles M.; NORTH, Michael J.: AGENT-BASED MODELING AND SIMULATION. *Simulation Conference (WSC), Proceedings of the 2009 Winter.* 2009, s. 86-98. DOI: 10.1109/WSC.2009.5429318

[12] MARIŠKA, Martin: *AGENTOVĚ ORIENTOVANÉ SIMULAČNÍ MODELY PROVOZU OBSLUŽNÝCH SYSTÉMŮ.* Pardubice, 2012, 90 s. Diplomová práce na Fakultě elektrotechniky a informatiky na Univerzitě Pardubice. Vedoucí diplomové práce prof. Ing. Antonín Kavička, Ph.D.

[13] KLÜGL, Franziska; HERRLER, Rainer; FEHLER, Manuel: SeSAm: Implementation of Agent-Based Simulation Using Visual Programming. *Proceedings of The 5th International Conference on Autonomous Agents and Multiagent Systems (AAMAS '06).* 2006, New York, USA, s. 1439-1440, ISBN:1-59593-303-4, DOI: 10.1145/1160633.1160904

[14] HIEBELER, David: *The Swarm Simulation System and Individual-based Modeling*.* 1994. DOI: 10.1.1.55.862

[15] OZIK, Jonathan: *ReLogoGettingStarted.* [online]. Repast Development Team URL: <http://repast.sourceforge.net/docs/ReLogoGettingStarted.pdf>, [cit. 17. 4. 2013]

[16] HUSÁKOVÁ, Martina: *Vývojové nástroje pro multiagentové systémy. Znalostní technologie III materiál pro podporu studia.* [online]. Univerzita Hradec Králové. Fakulta informatiky a managementu. Katedra informačních technologií na Univerzitě Hradec Králové. URL: <http://lide.uhk.cz/fim/ucitel/fshusam2/lekarnicky/zt3/zt3_dokumenty/MultiagentProst%C5%99ed%C3%AD.pdf>, [cit. 17. 3. 2013]

[17] BARTÁK, Roman: *Umělá inteligence I.* [online]. Univerzita Karlova. Matematicko-fyzikální fakulta. Katedra teoretické informatiky a matematické logiky. URL: <http://kti.mff.cuni.cz/~bartak/ui/lectures/lecture02.pdf>, [cit. 7. 4. 2013]

[18] JOHNSON, Paul; LANCASTER, Alex: *Swarm User Guide* [online]. 2000. URL: <http://pj.freefaculty.org/Swarm/Beta/SwarmUserGuide/userbook.html>, [cit. 17. 3. 2013]

[19] ALLAN, Rob: *Survey of Agent Based Modelling and Simulation Tools* [online]. Version 1.1 URL: <http://www.grids.ac.uk/Complex/ABMS>, [cit. 23. 2. 2013]

[20] *AgenBuilder* [online]. URL: <http://www.agentbuilder.com>, [cit. 26. 2. 2013]

[21] *Agent Foundation Classes (AFC)* [online]. URL: <http://www.cs.cmu.edu/~softagents/afc.html>, [cit. 3. 3. 2013]

[22] *AgentLink.org / European Co-ordination Action for Agent-Based Computing* [online]. URL: <http://www.agentlink.org>, [cit. 23. 2. 2013]

[23] *agentTool III* [online]. URL: <http://agenttool.cis.ksu.edu>, [cit. 24. 2. 2013]

[24] *COUGAAR* [online]. URL: <http://www.cougaar.org>, [cit. 2. 3. 2013]

[25] *The FIPA - IEEE Foundation for Intelligent Physical Agents* [online]. URL: <http://www.fipa.org/resources/livesystems.html>, [cit. 23. 2. 2013]

[26] *JADE* [online]. URL: <http://jade.tilab.com>, [cit. 23. 2. 2013]

[27] *MadKit* [online]. URL: <http://www.madkit.org>, [cit. 4. 3. 2013]

[28] *MASON* [online]. URL: <http://cs.gmu.edu/~eclab/projects/mason>, [cit. 9. 3. 2013]

[29] *Repast Suite Downloads* [online]. URL: <http://repast.sourceforge.net/download.html>, [cit. 23. 2. 2013]

[30] *RETSINA* [online]. URL: <http://www.cs.cmu.edu/~softagents/retsina_agent_arch.html>, [cit. 3. 3. 2013]

[31] *SeSAm* [online]. URL: <http://www.simsesam.de>, [cit. 10. 3. 2013]

[32] *SeSAm-Wiki* [online]. URL: <http://130.243.124.21/mediawiki/index.php/Main_Page>, [cit. 10. 3. 2013]

[33] *Swarm* [online]. URL: <http://www.swarm.org>, [cit. 17. 3. 2013]

SEZNAM PŘÍLOH

A OVĚŘENÍ SROZUMITELNOSTI UVEDENÝCH NÁVODŮ

Mým úkolem bylo srovnat a posoudit instalaci a spuštění dvou různých prostředí pro modelování SW agentů. Pro testování bylo zvoleno prostředí MASON, vyvinuté na Univerzitě George Masona, a Repast Simphony 2.0, vyvinuté v Argonské národní laboratoři.

Oba nástroje se spouští pod vývojovým prostředím Java SDK a jsou volně ke stažení z internetu. Při jejich stažení a instalaci jsem postupovala dle návodu uvedeného v této bakalářské práci.

Instalace a spuštění balíku MASON
Balík se povedlo úspěšně stáhnout z uvedené internetové adresy. Dále jsem postupovala dle návodu a spuštění ukázkové simulace proběhlo bez problému. I přes tuto skutečnost doporučuji zaměnit pořadí bodů 1. (spuštění Eclipse) a 2. (rozbalení archivních souborů).

Instalace a spuštění prostředí Repast Simphony 2.0
Prostředí bylo staženo z uvedené adresy. Součástí instalačního souboru je i prostředí Eclipse, proto je soubor poměrně velký. Jeho stažení může být v závislosti na rychlosti poskytovaného internetového připojení časově náročné. Doporučuji v návodu na tuto skutečnost upozornit. Další postup spuštění ukázkových příkladů proběhl v pořádku a příklady se zdařilo spustit.

Celkově hodnotím uvedené návody jako podrobné a velmi dobře popsané. Aplikací těchto návodů může uživatel nástroje bez větších problémů spustit včetně ukázkových příkladů.

Ing. Petra Slavíková
Ústav počítačové a řídicí techniky

Vysoká škola chemicko-technologická v Praze
Technická 5
166 28 Praha 6 – Dejvice
email: slavikop@vscht.cz

B ZDROJOVÝ KÓD JEDNODUCHÉ SIMULACE - MASON

V simulaci existuje jeden agent, který je pravidelně volán plánovačem. Agent v každém kroku vykoná pohyb náhodným směrem. Simulace je tvořena třemi třídami:

- **třída** *Tutorial* - modelová vrstva. Jedná se o potomka třídy *SimState*. Instance této třídy udržuje globální stav celé simulace, definuje prostředí a vytváří objekt agenta, který je naplánován k opakovanému spouštění.
- **třída** *Agent* - definuje chování agenta. Třída implementuje rozhraní *Steppable*, a proto musí implementovat metodu *step(SimState state)*, která je vyvolána vždy po naplánování agenta plánovačem. Vytvoření instance třídy *Stoppable* by umožnilo zrušit naplánování agenta zavoláním metody stop();
- **třída** *TutorialWithUI* - vizualizační vrstva. Jedná se o potomka třídy *GUIState*. Tato třída není nutná pro běh simulace. Slouží pro vizualizaci stavu objektů v modelové vrstvě.

```
import sim.engine.SimState;
import sim.field.grid.SparseGrid2D;
/**
 * Jednoduchá simulace s jedním agentem vytvořená v MASON
 * V simulaci existuje jeden agent, který je pravidelně volán plánovačem
 * Agent v každém kroku vykoná pohyb náhodným směrem

 * Třída Tutorial udržuje globální stav celé simulace, definuje prostředí a vytváří objekt agenta
 */
public class Tutorial extends SimState {
	// Dvourozměrné pole reprezentující prostředí
	// Interně používá hash tabulku, umožňuje uchovat více objektů ve stejné lokaci
	public SparseGrid2D environment;
	// Rozměry prostředí
	public int gridWidth = 50;
	public int gridHeight = 50;
	 /** Konstruktor
	 * @param seed - náhodné 32bitove číslo */
	public Tutorial(long seed) {
		super(seed);
	}
	/** Inicializace modelu
	 * @see sim.engine.SimState#start() */
	public void start() {
		// Reset a vyčištění plánováče
		super.start();
		// Vytvoření prostředí
		environment = new SparseGrid2D(gridWidth, gridHeight);
		// Vytvoření jednoho agenta
		Agent agent = new Agent();
		// Umístění agenta doprostřed prostředí
		environment.setObjectLocation(agent, gridWidth/2, gridWidth/2);
		// Naplánování agenta k opakovanému spouštění
```

```
                schedule.scheduleRepeating(agent);
        }
        /** Hlavní metoda
         * @param args vstupní parametry */
        public static void main(String[] args) {
                // Vytvoření instance podtřídy zděděné od SimState, předání vstupních parametrů
                // Inicializace generátoru nahodných čísel
                // Opakované volaní metody step(SimState state) diskrétním plánovačem
                doLoop(Tutorial.class, args);
                // Ukončení aplikace

import sim.engine.SimState;
import sim.engine.Steppable;
import sim.util.Int2D;
/**
 * Třída Agent - agent se při každém naplánování pohne náhodným směrem
 * Implementuje rozhraní Steppable -> musí obsahovat metodu step(SimState state)
 */
public class Agent implements Steppable {
        /** Metoda volaná při naplánování agenta plánovačem
         * @param state - stav modelu (SimState) */
        public void step(SimState state) {
                // Přetypování SimState na třídu Tutorial
                Tutorial tut = (Tutorial) state;
                // Zjištění umístění agenta v prostoru
                Int2D location = tut.environment.getObjectLocation(this);
                // Náhodný pohyb agenta
                int newx = location.x + tut.random.nextInt(3) - 1;
                int newy = location.y + tut.random.nextInt(3) - 1;
                // Agent se při dosažení hranic prostředí objeví na druhé straně
                if (newx < 0) { newx = tut.environment.getWidth() - 1; }
                else if (newx >= tut.environment.getWidth()) { newx = 0; }
                if (newy < 0) { newy = tut.environment.getHeight() - 1; }
                else if (newy >= tut.environment.getHeight()) { newy = 0; }
                // Nastavení nové pozice agenta v prostředí
                tut.environment.setObjectLocation(this, new Int2D(newx, newy));
        }
        /** Metodu getType() volá inspektor
         * @return informace o agentovi */
        public String getType() {
                return "Jsem agent";
        }
}

import java.awt.Color;
import javax.swing.JFrame;
import sim.display.Controller;
import sim.display.Display2D;
import sim.display.GUIState;
import sim.engine.SimState;
import sim.portrayal.grid.SparseGridPortrayal2D;
/**
 * Třída pro vizualizaci simulace (potomek třídy GUIState)
 */
public class TutorialWithUI extends GUIState {
        // Dvourozměrný objekt umožňující vizualizaci (komponenta Swing JComponent)
        public Display2D display;
        // Okno pro zobrazení (Objekt JFrame)
        public JFrame displayFrame;
```

```
// Dvorozměrné pole zodpovědné za vykreslování a dovolující inspekci pole
SparseGridPortrayal2D gridPortrayal = new SparseGridPortrayal2D();
/** Konstruktor */
public TutorialWithUI() {
	super(new Tutorial(System.currentTimeMillis()));
}
/** Konstruktor
 * @param state - stav modelu (SimState) */
public TutorialWithUI(SimState state) {
	super(state);
}
/** Název okna
 * @return popis simulace */
public static String getName() {
	return "Tutorial: vytvoření jednoduché simulace";
}
/** Metoda start je zavolána po stisku tlačítka Play v rámci konzole */
public void start() {
	super.start();
	// Nastavení vizualizace
	setupPortrayals();
}
/** Metoda je zavolána při obnovování simulace z checkpointu */
public void load(SimState state) {
	super.load(state);
	// Nastavení vizualizace
	setupPortrayals();
}
/** Nastavení vizualizace */
public void setupPortrayals() {
	// Nastavení pole potrayal, které pole má zobrazovat
	gridPortrayal.setField(((Tutorial) state).environment);
	// Nastavení pole potrayal, jak má zobrazovat objekty
	gridPortrayal.setPortrayalForAll(new sim.portrayal.simple.OvalPortrayal2D(Color.green)
	    );
	// Reset displeje
	display.reset();
	// Překreslení displeje do inicializačního uspořádání
	display.repaint();
}
/** Grafická konzole pomocí níž lze ovládat simulaci */
public void init(Controller c) {
	// Inicializace
	super.init(c);
	// Vytvoří displej o velikosti 400x400
	display = new Display2D(400, 400, this, 1);
	// Vytvoření snímku
	displayFrame = display.createFrame();
	// Registrace displeje (Display2D) do seznamu JFrames kontroleru
	// To umožní ovládat displej v konzoli (schovávat, obnovovat)
	// Zajistí tako to, že je-li model změněn graficky uživatelem, displej je aktualizován
	c.registerFrame(displayFrame);
	// Připojení portrayals k displeji
	display.attach(gridPortrayal, "Agent");
	// Nastavení, že je frame viditelný
	displayFrame.setVisible(true);
	// Nastavení pozadí - co má být za displejem
	display.setBackdrop(Color.black);
}
```

```
        /** Metoda je zavolána po zavření grafické konzole */
        public void quit() {
                super.quit();
                // Zajištění, že frame bude zavřen pouze jednou
                if (displayFrame != null) displayFrame.dispose();
                // Pomoc pro garbage collection
                displayFrame = null;
                display = null;
        }
        /** Hlavní metoda
         * @param args vstupní parametry */
        public static void main(String[] args) {
                new TutorialWithUI().createController();
        }
}
```

C ZDROJOVÝ KÓD JEDNODUCHÉ SIMULACE - REPAST SIMPHONY 2.0

V simulaci existuje jeden agent, který je pravidelně volán plánovačem. Agent v každém kroku vykoná pohyb náhodným směrem. Simulace je tvořena třemi třídami, jejichž výčet je vidět níže. Model simulace se částečně definuje pomocí parametrů uložených v XML souborech. Jsou mezi nimi i parametry pro nastavení rozměrů prostředí.

- **třída** *UserObserver* - modelová vrstva. Jedná se o potomka třídy *BaseTurtle*;
- **třída** *UserTurtle* - definuje chování agenta. Jedná se o potomka třídy *BaseObserver*. Třída *UserTurtle* musí implementovat metodu *step()*, která je vždy vyvolána po naplánování agenta plánovačem;
- **třída** *UserGlobalsAndPanelFactory* - obsahuje uživatelem nadefinované prvky pro ovládání simulace. Jedná se o potomka třídy *AbstractReLogoGlobalsAndPanelFactory*.

```
import static repast.simphony.relogo.Utility.*;
import static repast.simphony.relogo.UtilityG.*;
import repast.simphony.relogo.BaseObserver;
import repast.simphony.relogo.Stop;
import repast.simphony.relogo.Utility;
import repast.simphony.relogo.UtilityG;
/**
 * Jednoduchá simulace s jedním agentem vytvořená v Repast Simphony 2.0
 * V simulaci existuje jeden agent, který je pravidelně volán plánovačem
 * Agent v každém kroku vykoná pohyb náhodným směrem
 *
 * Třída UserObserver obsahuje definici modelu (potomek třídy BaseObserver)
 */
class UserObserver extends BaseObserver{
	/** Inicializace modelu */
	def setup() {
		// Resetu do počátečního stavu
		clearAll()
		// Nastavení tvaru objektu UserTurtle na kolečko
		setDefaultShape(UserTurtle, "circle")
		// Vytvoření objektu jednoho objektu UserTurle
		createUserTurtles(1) {
			// Nastavení barvy objektu na zelenou
			setColor(green())
			}
	}
	/** Definice kroku simulace */
	def go(){
		// V každém kroku se zavolá metoda step() objektu userTurtles
		ask (userTurtles()){ step() }

import static repast.simphony.relogo.Utility.*;
import static repast.simphony.relogo.UtilityG.*;
import repast.simphony.relogo.BaseTurtle;
```

```
import repast.simphony.relogo.BasePatch;
import repast.simphony.relogo.Plural;
import repast.simphony.relogo.Stop;
import repast.simphony.relogo.Utility;
import repast.simphony.relogo.UtilityG;
/**
 * Třída UserTurtle - agent se při každém naplánování pohne náhodným směrem
 * Je potomkem třídy BaseTurtle -> musí obsahovat metodu step()
 */
class UserTurtle extends BaseTurtle{
	/** Metoda volaná při naplánování agenta plánovačem */
	def step(){
		// Zjistění umístění agenta v prostoru a náhodná změna souřadnic
		double newx = getXcor() + random(3) - 1
		double newy = getYcor() + random(3) - 1
		// Nastavení nové pozice agenta v prostředí
		setxy(newx, newy)
	}
}

import repast.simphony.relogo.factories.AbstractReLogoGlobalsAndPanelFactory
/**
 * Třída UserGlobalsAndPanelFactory obsahuje ovládací prvky simulace
 * Pomocí těchto pruků je možné ovládat běh simulace
 */
public class UserGlobalsAndPanelFactory extends AbstractReLogoGlobalsAndPanelFactory{
	/** Metoda s definicí ovládacích pruků */
	public void addGlobalsAndPanelComponents(){
		// Tlačítko pro nastavení modelu
		addButtonWL ("setup","Setup")
		// Tlačítko pro krokování běhu modelu
		addButtonWL ("go","Go␣Once")
		// Tlačítko pro spuštění běhu modelu
		addToggleButtonWL ("go","Go")
	}
}
```

D OBSAH PŘILOŽENÉHO CD

K bakalářské práci je přiložené CD, které obsahuje instalační soubory nástrojů MASON a Repast Simphony 2.0. Dále obsahuje zdrojové kódy modelů simulací, jejich spustitelné verze a návody pro spuštění. Na CD je uloženo i celé znění bakalářské práce včetně zdrojového textu.

Adresářová struktura (uvedeny pouze podstatné soubory a adresáře):

```
Instalační soubory
   - MASON
   - Repast Simphony 2.0
Simulační modely
   - MASON
      - Spustitelná verze
         -INFO.TXT
      - Zdrojové kódy
   - Repast Simphony 2.0
      - Spustitelná verze
         -INFO.TXT
      - Zdrojové kódy
Text bakalářské práce
   - Zdrojový text
   - PDF
```