UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

BAKALÁŘSKÁ PRÁCE

2008 **Dvorský Tadeáš**

UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Podpora vizualizace agentů v ABAsim architektuře simulačních modelů pomocí knihovny NetBeans Visual Library

BAKALÁŘSKÁ PRÁCE

AUTOR PRÁCE: **Dvorský Tadeáš**
VEDOUCÍ PRÁCE: **Ing. Jiří Pinkas**

2008

UNIVERSITY OF PARDUBICE
FACULTY OF ELEKTRICAL ENGINEERING AND INFORMATICS

Support vizualization of agents based on ABAsim architetucture of simulation models using Netbeans Visual Library

BACHELOR WORK

AUTHOR: **Dvorský Tadeáš**
SUPERVISOR: **Ing. Jiří Pinkas**

2008

Univerzita Pardubice
Rektorát
Fakulta elektrotechniky a informatiky
Akademický rok: **2007/2008**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Tadeáš DVORSKÝ**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Podpora vizualizace agentů v ABAsim architektuře simulačních modelů pomocí knihovny NetBeans visual library**

Zásady pro vypracování:

Pomocí knihovny NetBeans visual library implementovat ve vyšším programovacím jazyku Java sadu grafických komponent pro vizualizaci agentů v ABAsim architektuře simulačních modelů.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] KAVIČKA, A., KLIMA, V., ADAMKO N.: Agentovo orientovaná simulacia dopravných uzlov. Žilina: EDIS, 2005. 206 s. ISBN: 80-8070-477-5

[2] NetBeans Visual Library [online]. Dostupný z WWW: <http://graph.netbeans.org/>

[3] BOUDREAU, T., TULACH, J., WIELENGA, G.,: Rich Client Programming: Plugging into the NetBeans(TM) Platform. Prentice Hall, 2007. 640 s. ISBN: 01-3235-4802

[4] Online zdroje na Internetu

Vedoucí bakalářské práce: **Ing. Jiří Pinkas**
Katedra informatiky v dopravě

Datum zadání bakalářské práce: **30. listopadu 2007**

Termín odevzdání bakalářské práce: **16. května 2008**

Univerzita Pardubice
Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
rektor

V Pardubicích dne 29. dubna 2008

Poděkování:

Rád bych poděkoval Ing. Jiřímu Pinkasovi za odborné rady a vedení při zpracování bakalářské práce.

Abstrakt

Tato práce porovnává návrhy vizualizace agentů ve vrstvě managementu v modelu ABAsim a řešení možné komunikace mezi nimi. V první části této práce je popsáno co to je agent, princip na jakém funguje, jeho použití v multiagentovém systému, druhy komunikace a její použití. Jsou zde také uvedeny jednotlivé návrhy řešení.

V implementační části je potom na základě konkrétního vybraného návrhu popsána jeho realizace pomoci NetBeans.

Klíčová slova

Agent, komunikace, multiagentový systém, ABAsim, NetBeans

Abstract

This work compares proposals of agent visualization in ABAsim architecture of simulation models and their communication. The first part of this work describes an agent, it's usage in multiagent systems, types of communication and their usage. In following chapters is described current agent visualisation and proposals for alternative visualisations.

In second part of this work is chosen one proposal and described it's implementation using NetBeans Visual Library.

Keywords

Agent, communication, multiagent system, ABAsim, NetBeans.

Obsah

Seznam obrázků

Seznam použitých zkratek

ABAsim	**A**gent **B**ased **A**rchitekture of **sim**ulation model. Agentově založená archytektura simulačních modelů.
ACL	**A**gent **C**ommunication **L**anguage Jazyk pro komunikaci agentů
CDDL	**C**ommon **D**evelopment and **D**istribution **L**icense
FIPA	**F**oundation for **I**ntelligent **P**hysical **A**gent
KQML	**K**nowledge **Q**uery **M**anipulation **L**anguage Jazyk pro komunikaci agentů, vychází z ACL.
MAML	**M**ulti **A**gent **M**odeling **L**anguage
MAS	**M**ulti-**A**gent **S**ystems,. Multiagentový systém.
NVL	**N**etBeans **V**isual **L**ibrary
XML	e**X**tensible **M**arkup **L**anguage Rozšiřitelný značkovací jazyk.

1 Úvod

Tato práce se zaměřuje na možné úpravy současné vizualizace agentů a jejich komunikace. Stávající forma vizualizace agentů je při větším množství agentů a komunikace mezi nimi značně nepřehledná. Za léta vývoje agentů by se mohlo zdát, že se ustálily pojmy a definice stejně jako to jak zachytit agentovou komunikaci, ale i v současné době se ještě pojmy neustálily. Nejprve se tato práce zaměřuje na možnosti vycházející ze současného modelu vizualizace. Další možnosti, které jsou zde popsány jsou nové, či odlišné od stávajícího modelu. Poté je zde uvedena implementace a popis vybrané vizualizace na prototypu aplikace postavené na management vrstvě ABAsim modelu. V současné době existuje celá řada programů pro tvorbu a vývoj agentů, ale převládá v nich klasický způsob vizualizace pomocí obdélníků či kruhů pospojovaných čárami. Tato práce má za cíl navrhnout přehlednější způsob komunikace a správu samotných agentů.

Text práce je rozdělen na dvě části. V první části je uvedena agentová teorie, ze které tato práce vychází, jsou zde vymezené základní definice agenta, jaké jsou druhy agentů, jaké má agent vlastnosti apod. Druhá část je věnována prostředí, ve kterém je aplikace vybudována. Jsou zde podrobněji popsány prvky použité v aplikaci, jež toto prostředí nabízí. Na konci práce je nastíněn další možný vývoj v této oblasti.

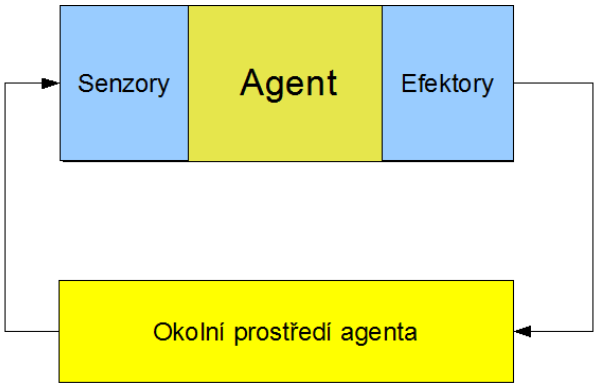
2 Agentově orientované programování

2.1 Agent

Pojem agent se v počítačové literatuře objevil poprvé někdy v 60. letech minulého století. Agent je v podstatě autonomní systém, schopný vnímat své okolí, komunikovat s ním, ale také je schopen samostatného rozhodování o svých příštích akcích. Od té doby vývoj a teorie agentů pokročily a v současné době se spíše používají multiagentové systémy než samotní agenti.

Agentem tedy může být cokoliv, co vnímá a reaguje na své okolí. Existuje celá řada agentů nejen těch počítačových např. biologičtí agenti jsou lidé, techničtí agenti – roboti, počítačoví agenti se dělí na agenty v počítačových hrách, počítačové viry, specifické agenty určené pro nějakou úlohu a jako entity umělého života. Počítačový agent je vlastně zapouzdřený systém, ale k jeho realizaci jsou nutné hardwarové komponenty. Agent má senzory, to jsou snímače jeho okolí pomocí nich

dostává zprávy (z prostředí, ve kterém se vyskytuje, mohou to být jiní agenti) na své okolí pak prostřednictvím efektorů (obr. 1). Agent je schopen samostatně rozhodovat o akcích, které uskutečňuje pro dosažení daného cíle.



Obr. 1: Interakce agenta a okolního prostředí (1)

2.2 Vlastnosti agenta

Mezi klíčové vlastnosti agenta patří:

- *Autonomie*, znamená jeli agent cílově orientovaný modul, schopný samostatného řešení určitých úloh bez nezbytné komunikace s okolím, avšak schopní komunikace, koordinace činnosti či kooperace s jinými agenty v rámci určité komunity. Mají možnost dobrovolně vstupovat a opouštět komunitu, poskytovat či požadovat výsledky.
- *Reaktivita*, znamená to, že je agent schopen adekvátní reakce na změny ve svém okolí.
- *Společenské chování*, agenta spočívá v tom, že je agent schopen komunikace s jiným agentem nebo člověkem pomocí nějaké formy komunikace.
- *Iniciativnost,* znamená to, že agent by měl mít schopnost reakce na změny ve svém okolí, ale měl by mít i chování zaměřené na dosažení cíle, který ovlivňuje akce agenta.

Samotná technická realizace agenta vychází z toho na jakou jeho vlastnost je kladen důraz a s tím souvisí i agentova samotná implementace v určeném prostředí. Jednotlivé druhy přístupu realizace agentů vychází z různých problematik v informačních technologií nebo potřeb jednotlivých aplikací, ve kterých mají být agenti použiti.

2.3 Druhy agentů

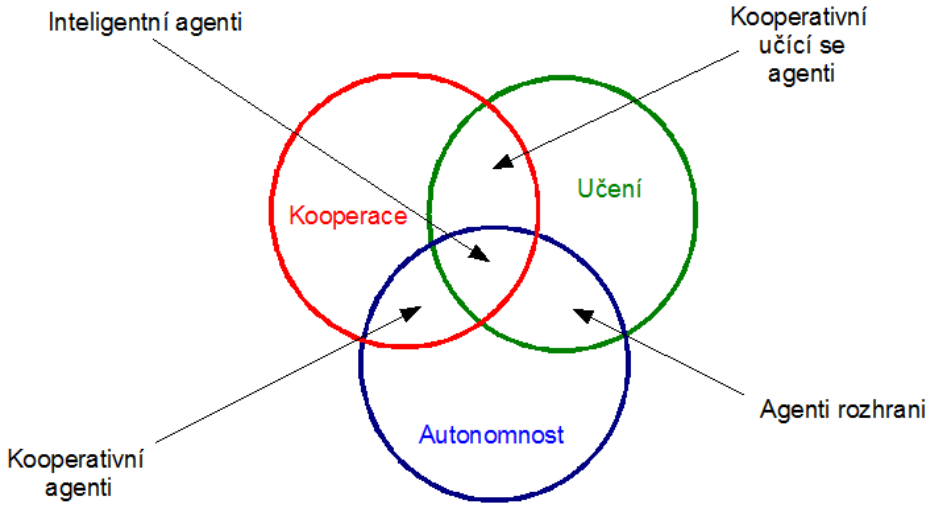
Problematika klasifikace druhů agentů není v současné době zatím ještě úplně ustálena. Základní klasifikace agentů, kteří mohou být členění dle různých kritérií a to:

- Prvním kritériem je *mobilita* agentů. Je to schopnost jejich pohybu v prostředí. Dělí se na dva druhy, prvním druhem jsou statičtí agenti tito agenti se nepřemísťují, ale setrvávají na jednom místě v rámci prostředí nebo v počítačové síti na jednom počítači. Oproti tomu mobilní agenti se mohou pohybovat.

- Druhým kritériem je *míra iniciativnosti agenta*, přičemž rozlišujeme uvažujících a reaktivních agentů. *Rozhodující* se agenti jsou založeni na logickém uvažování nebo v sobě zahrnují model okolního prostředí, jsou tedy schopni kdykoliv rozhodnout o tom jakou činnost v daném momentu vykonávat. *Reaktivní* agenti fungují na principu podnět→odezva, jejich rozhodnutí tedy vyplývá z aktuálního stavu prostředí v němž jsou umístěni. Reaktivní agenti tedy pouze reagují na podněty z okolí.

- Třetí možné rozdělení je podle toho na co se agent zaměřuje, tedy na jeho hlavní vlastnosti *autonomnost*, *kooperaci* a *schopnost učit se*. Rozlišujeme tedy kooperativní agenty, agenty rozhraní, kooperativní učící se agenty a inteligentní agenty.

Na obrázku (Obr. 2) jsou znázorněné vzájemné průniky dle uvedených vlastnosti agentů.



Obr. 2: Klasifikace agentů (2)

Kromě těchto základních uvedených členění je možné dále klasifikovat podle jejich konkrétního aplikačního poslání (například *internetový agent* působí v prostředí internetu, *informační agent* pracuje ve spojení s určitým informačním systémem atd.) anebo podle použitého koncepčního přístupu (filozofie), který je při konstrukci agenta uplatněný. V případě, že je při tvorbě jednotlivého agenta použita kombinace vícerých koncepčních přístupů, které byli diskutované, tak hovoříme o *hybridním agentovi* (ten může například kombinovaně využívat principy uvažujících a reaktivních agentů apod.)[1]

Existuje i důsledná klasifikace jež určí přesné začlenění agenta. Ja zde však uvedu pouze jednodušší klasifikaci zdůrazňující nejvíce používané typy agentů.

Agenti se tedy člení na:

- Kooperativní agenty.
- Agenty rozhraní.
- Mobilní agenty.
- Informační/internetové agenty.
- Reaktivní agenty.
- Hybridní agenty.
- Inteligentní agenty.

V případě,že aplikace kombinuje dvou nebo více z výše uvedených druhů agentů říkáme, že se jedná o heterogenní (různorodém) systém agentů.

2.3.1 Popis vybraných architektur agentů

Inteligentní agent navozuje představu, že agent řeší úlohy inteligentním způsobem. Tím se myslí agentova schopnost plnit cíle, které jsou v jeho zájmu pomocí jeho vlastní „inteligence“, ve většině případů logickou dedukcí. Agentovy záměry většinou souvisí s účelem jeho vzniku. Agent je však nemusí mít vždy pevně zabudované, ale může je také získávat od jiných agentů a pak se jimi řídit.

Uvažující agent má na rozdíl od reaktivního agenta schopnost plánovat postup svých akcí vedoucích k dosažení zvolených nebo zadaných cílů. To znamená, že agent musí mít schopnost různých výpočtů, což může být druh vnitřní činnosti agen-

ta. K dosažení svých záměrů pak agent ovlivňuje okolní prostředí tak, aby získal nějakou výhodu. Toto jednání je další z často uváděných vlastností agentů a nazývá se *proaktivita*.

Kognitivní agent vyvozuje logické závěry z pozorování okolního prostředí. Takový agent se především učí a vytváří svou vlastní základnu znalostí. Do ní si během svého působení ukládá informace získané interakcí s okolím nebo znalosti získané dedukcí. Kognitivní agent nemusí být založen nutně na uvažujících schopnostech, potom provádí pouze vnitřní akce, například analyzuje scénu, provádí překlad, nebo získává znalosti .

Racionální agent má všechny výše uvedené vlastnosti (i vlastností reaktivního agenta, který je uveden níže) 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.

Mobilní agenti jsou založeni na myšlence autonomního transferu zdrojového kódu agenta na jinou operačně-hardwarovou platformu a pokračování jeho vykonávání kódu na novém místě. V některých případech implementace jde o alternativu architektury klient-server, ve které se síťovou infrastrukturou přenáší požadavky a data. V případě mobilních agentů se přenáší funkcionalita, která v mnohých případech představuje stručný kód nezatěžující přenosové linky. Procesy probíhající od vytvoření instance agenta až po jeho ukončení jsou uzavřeny ve smyčce. Životní cyklus mobilních agentů je perzistentní nebo úkolový. V prvním případě agent po přenosu pokračuje ve vykonávání kódu v bodě, kde byl běh přerušen, v druhém je vykonávání obnoveno bez „pamatování" předchozího stavu nebo vykonaných metod. S přenosem agentů počítačovou sítí jsou spojeny problémy bezpečnosti a správy.

Kooperativní agent má za cíl koordinaci i kooperaci. Řídí a rozděluje úlohy mezi agenty celého systému. Kooperativní agent vnáší řád do jednání společenství agentů a tím vytváří společenské normy systému. Metody kooperace přinášejí principy organizace činnosti skupiny agentů realizujících danou úlohu. Koordinace a kooperace spolu těsně souvisí – rozhodnutí v jedné oblasti vyžadují určité postupy v oblasti druhé a naopak.

Agenti rozhraní mají za cíl spravovat různé typy rozhraní. Jednou ze zásadních rysů inteligentních prostředí je proměna uživatelských rozhraní z dnes zatím obvykle používaných displejů na vysoce intuitivně využitelná rozhraní ovládána hlasem, dotykem, apod. Pro takovéto typy uživatelských rozhraní může být využití technologie agentů velkým přínosem. Využití agentových přístupů je tedy v tomto směru velikou výzvou.

Reaktivní agent vždy pouze reaguje na podněty z vnějšího světa. Zná pouze omezenou množinu akcí, na něž je schopen reagovat. Upřesňuje svůj obraz světa na základě vnějších podnětů. Své výsledky může sdělovat ostatním. Za jistých okolností mu hrozí zacyklení. Reaktivní agenti nedisponují interním modelem svého okolního prostředí. Racionalita jeho chování je důsledkem interakce s prostředím. Nemá žádný modul pro tvorbu plánů ani modul pro rozhodování, který cíl z potencionální množiny cílů bude sledovat. Jeho součástí jsou tzv. kompetenční moduly, což jsou funkce nebo procedury, které se vykonávají na základě splnění aktivační podmínky.

Při konstrukci reaktivního agenta se neuvažuje o vytvořeni nějakého plánu agenta. Reaktivní agent bývá označován jako situační protože nemá vytvořené žádné plány jen reaguje na to co se právě děje.

Konstruovaný reaktivní agent se skládá z modulů, které pracují autonomně a jsou zodpovědné za specifické úlohy. Mezi moduly existuje minimální komunikace. V rámci žádného z agentů neexistuje žádný interní model celého systému, a proto celková správa agenta vyplyne až z jeho působení uvnitř většího celku.

Působení reaktivních agentů připomíná více činnost senzorických systémů, než činnost systémů s prvky umělé inteligence.

Reprezentace reaktivního agenta, jako i jeho interakce s ostatními agenty jsou relativně velmi jednoduché. Celkový charakter spravování individuálního agenta vyplývá až z jeho působení v rámci celých společenství agentů, které jsou sledované jako celek.

Inteligentní agentový systém může být postavený jen na reaktivních agentech a inteligence tohoto systému se projeví, až v důsledku interakcí všech agentů v tomto systému.

2.4 Prostředí

Prostředí je vše, s čím agent přichází během své činnosti do styku. Je to tedy agentní systém bez jediného svého prvku - agenta. Prostředí z hlediska agenta pak může být.

- *Plně pozorovatelné/Částečně pozorovatelné.* Prostředí je pro agenta plně pozorovatelné, pokud může svými senzory sledovat jeho kompletní stav.
- *Statické/Dynamické*. Prostředí je pro agenta statické, pokud se může měnit pouze jeho akcemi.
- *Deterministické/Nedeterministické.* Prostředí je deterministické, jestliže je jeho stav po vykonání nějaké akce dán pouze touto akcí a předcházejícím (původním) stavem tohoto prostředí.
- *Diskrétní/Spojité*. Prostředí je diskrétní, pokud má konečně nebo spočetně mnoho stavů. Pro agenta založeného na číslicových počítačích/procesorech je každé prostředí diskrétní. Údaje ze senzorů se totiž ukládají do konečného binárního řetězce a proto existuje pouze konečná množina stavů tohoto prostředí. [2]

2.5 Agenti v simulacích

Co vlastně přináší uplatnění agentů v oblasti simulací? Za prvé to je vysoký stupeň decentralizace z toho vyplývá, že není potřeba tvořit žádný globální model řízení celého společenství agentů, neboť každý z nich je autonomní. Za druhé je to vysoká životaschopnost systému.

Uvedené rysy by mohly být poměrně zajímavé pro tvorbu komplexních simulačních modelů (např. území rozsáhlých obslužných systémů a nebo biologických společenstev), pro které by bylo možné navrhnout dobře udržovatelnou a srozumitelnou strukturu členěnou například na přirozené autonomní jednotky řazení (např. při složitých obslužných systémech) a nebo na jednotlivé autonomní jedince (např. u biologických společenstev). Právě agenti se jeví jako dobří kandidáti na realizaci takovýchto autonomních řídících jednotek, resp. autonomních jedinců.[1]

Pokud již existuje implementačně-technická podpora třeba specializovaný software pro tvorbu *agentově-orientovaného programování* je možné ji použít. Pokud není dostupná nebo nevyhovuje daným potřebám, pak nezbývá, nežli navrhnout a realizovat vlastní. Zde se řeší především to jakým způsobem mají být implementováni jednotliví agenti, jak má být zabezpečený mechanizmus jejich komunikace a jaká koncepce bude použita na synchronizaci simulačního výpočtu.

2.6 Agentově orientovaná architektura simulačního modelu

Zatím pojem architektura simulačního modelu je dosti volné téma, proto uvedu alespoň dva základní prvky, na kterých tato architektura stojí.

Za prvé to jsou základní prvky. Základní strukturální a nebo funkčně-akční jednotky, s jejích pomocí je simulační model postavený (např. *událost, aktivita, proces, agent* apod.),

Za druhé je to rozdělení simulačního výpočtu na výpočtové jednotky. Z tohoto hlediska existují dva druhy. Pokud je simulační model realizován na jednoprocesorovém počítači jedná se o monolitickou architekturu. V opačném případě se jedná o paralelní (distribuovanou) architekturu, která je realizována na víceprocesorovém počítači,nebo v počítačové síti.

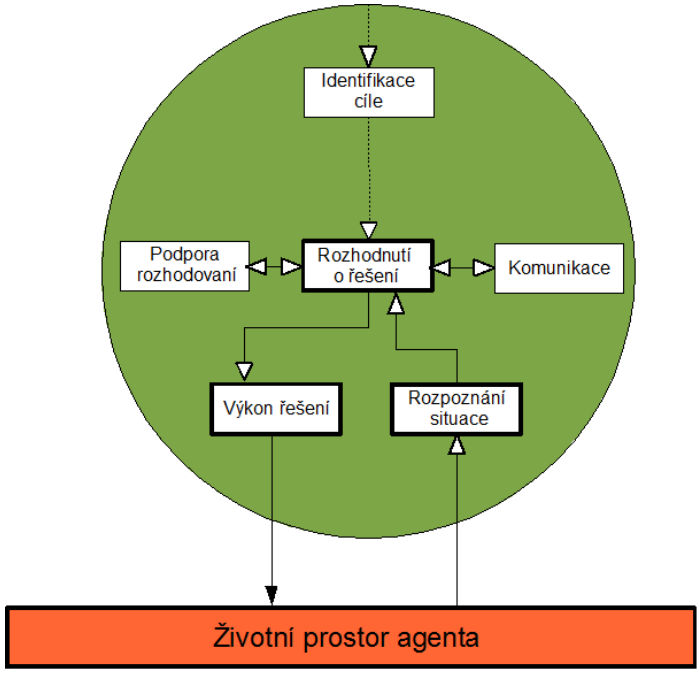
Každý typ architektury má možnost různým způsobem implementovat synchronizaci simulačního výpočtu.

Následuje popis specifických funkcí agenta v architektuře simulačního modelu inspirovaného filozofii reaktivních agentů (Obr. 3).

Každý agent má zadán určitý *cíl,* jež má plnit. Tento cíl mu je buď zadán, nebo v případě inteligentních agentů si jej identifikuje sám. V souladu s tím pro co je agent zkonstruován realizuje svůj životní cyklus a to následovně: *rozpoznání situace − rozhodování o řešení − výkon řešení*. Život a tedy rozhodování agenta je dáno také tím v jakém se nachází prostředí, to zahrnuje všechny ostatní agenty a také tu část prostoru simulačního modelu, ve kterém je agent oprávněný vykonávat změny. Prostředí působení agenta může být buď vymezené nebo v případě mobilních agentů se může dle potřeby měnit.

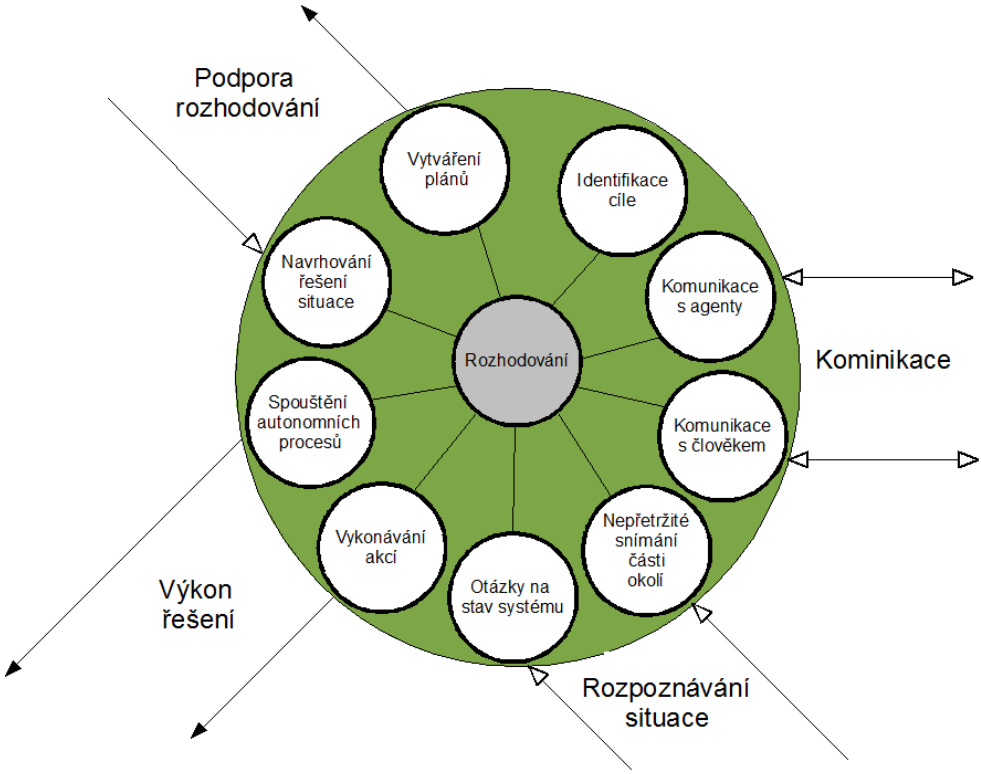
Pokud se agent rozhoduje, pak používá funkci *návrhu řešení problémů* a *komunikace* s jinými agenty. Dostane-li agent podnět a řešení dané situace, které je

v jeho kompetenci, pak pomáhá řešit problém jiným agentům tím, že je na tuto skutečnost upozorní.



Obr. 3: Funkce agenta(2)

K tomu, aby mohl agent efektivně vykonávat výše uvedené funkce je nutné, aby disponoval jistými schopnostmi (Obr. 4).



Obr. 4: Schopnosti agenta (2)

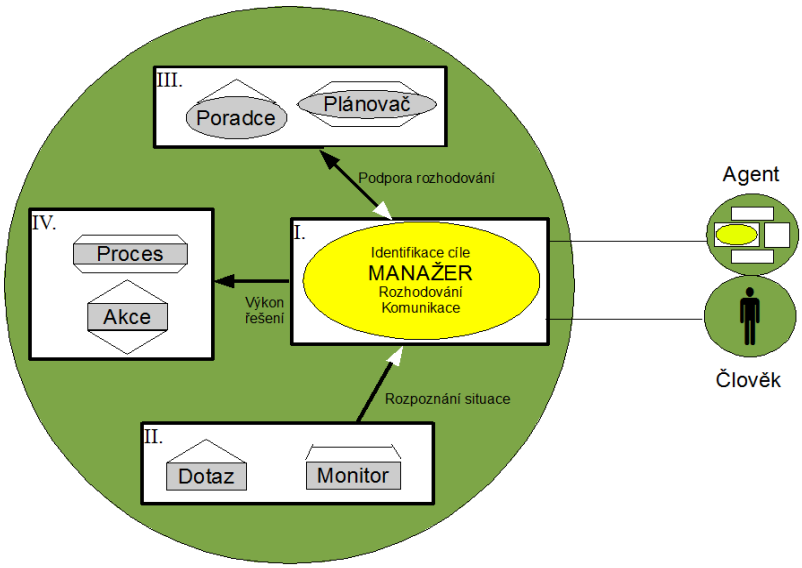
Pro komunikaci agentů s jinými agenty nebo s člověkem mu slouží komunikační mechanizmus. Agent má schopnost navrhovat jednorázové řešení problémů např. volání optimalizačních algoritmů, dále má schopnost sestavovat dlouhodobější plány k vykonávání obslužných činností jako je např. rozvržení zdrojů. Tato činnost však není vyjádření o tom jak se bude chovat agent v budoucnosti. Senzorická činnost umožňuje agentovi se ptát na stav systému či snímat části svého okolního prostředí. Svou výkonnou funkcí realizuje buď aktivací akce, která okamžitě mění stav systému nebo spustí proces, který mění stav systému v určitém časovém intervalu v jistých časových okamžicích bez jakýchkoliv vnějších zásahů.

Schopnosti agenta je vhodné rozdělit do skupin a každou takovouto schopnost implementovat jako samostatnou interní komponentu. Hlavními důvody pro tuto dekompozici agenta jsou:

- Možnost případného sdílení komponenty vícero agenty a její opakované použití.
- Možnost implementace vícerých alternativních verzí jedné komponenty, ze které si uživatel vybere jeden vhodný při sestavování scénáře simulačního experimentu.[1]

2.7 Dekompozice agenta

Agenta je tedy vhodné rozložit a to například na čtyři skupiny interních komponentů (Obr. 5).



Obr. 5: Dekompozice agenta (2)

I. Je to skupina *řídících a rozhodovacích komponent.* Zde je zařazená komponenta *manažer*. Tato komponenta vybírá ze schopností agenta schopnost identifikovat cíl, schopnost komunikovat a rozhodovat se. *Manažer* je pro agenta klíčový v tom smyslu, že jako jediná jeho komponenta umí komunikovat s ostatními interními komponenty, které mezi sebou nijak nekomunikují.

II. Skupina jež má na starost zpřístupnění informací o stavu systému nazývá se *skupinou senzorů*, může obsahovat komponenty dvou druhů. *Dotaz* je vyslán na pokyn manažera okamžitě za účelem zprostředkování informací. *Monitor* vykonává dlouhodobější snímání určité části v prostoru simulačního modelu a podává manažerovi jenom ty informace, které jsou pro něj významné.

III. Skupina *řešitelů* pomáhá manažerovi v rozhodování tím, že mu poskytuje možné návrhy řešení problémů. *Poradce* je pasivní komponenta, která na pokyn manažera okamžitě navrhne způsob řešení problémů. Poradcem je obvykle optimalizační algoritmus nebo člověk.

IV. Poslední skupinou komponent jsou *efektory.* Vykonávají rozhodnutí manažerů a mění tak stav systému. Žádné jiné komponenty stav systému měnit nemohou. *Akce* zabezpečí jednorázovou a okamžitou změnu stavu, zatím co *proces* po svém spuštění autonomně mění hodnoty stavových proměnných v jistých časových okamžicích.

Pro úplnost existuje ještě jiný způsob. Není tak používaný a je založen na předcházení konfliktů. V předstihu pro jistý časový okamžik je navrhnuto několik způsobů řešení ve formě časových plánů. O vytváření těchto plánů se stará komponenta plánovač. Plánovač se ve zvolených časových okamžicích sám aktualizuje bez iniciativy manažera. Pokud plánovač objeví vážný problém uvědomí o tom manažera.

Efektory, *senzory* a řešitelé se souhrnně nazývají asistenti, přičemž je možné je dělit na *kontinuální asistenty*, jejich činnost trvá nenulový simulační čas a *promptní asistenty*, jejichž činnost se vykonává v jednom okamžiku simulačního času (akce, dotazy a poradce).

Architekturu simulačního modelu založenou na uvedené koncepci (reaktivních) agentů pracovně označujeme jako architekturu *ABAsim* (**A**gent-**B**ased **A**rchitekture of **sim**ulation model).[1]

2.8 Multiagentový přístup

V nejjednodušším případě modelování reálného světa můžeme pracovat i jen s jediným agentem, popisujícím jediný systém. Nicméně běžnější je situace, kdy reálný svět modelujeme prostřednictvím sady agentů, modelujících různé relativně samostatné podsystémy a také relace a interakce mezi nimi. V takovém případě mluvíme o *multiagentových systémech*.

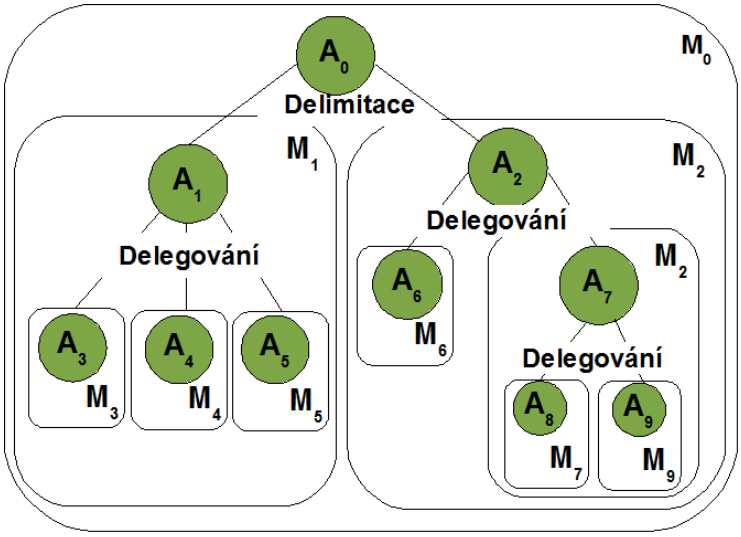
Teorie multiagentových systémů je poměrně mladou vědeckou oblastí zatím v ní nejsou ustálená jasná pravidla. Multiagentový systém je tedy tvořen skupinou agentů, kteří představují aktivní prvky, dále zde mohou být objekty, které žádnou aktivitu nevyvíjejí, ale svojí existencí ovlivňují chování agentů. Toto vše je zasazeno do určitého prostředí, které zprostředkovává interakci mezi agenty i objekty. Prostředí má rovněž vliv na chování agentů, respektive je jimi ovlivňováno. Prostředí rovněž svými vlastnostmi vymezuje možnou interakci mezi agenty navzájem stejně jako interakci agentů s objekty. Agenti a objekty zde vstupují do celé řady vzájemných vztahů. Multiagentové systémy většinou pracují v diskrétních časových krocích.

Za multiagentový lze proto považovat takový systém, který se skládá z následujících komponent:

- *Prostředí E*, jehož jednou z vlastností může být i prostorovost, tj. může se jednat o prostor, který je obecně n-rozměrný.
- *Množina objektů O*. Tyto objekty jsou situované, tj. v kterémkoliv okamžiku je možné k objektu přiřadit polohu v *E*. Objekty jsou zpravidla pasivní, mohou být vnímány, vytvářeny, odstraňovány a modifikovány agenty.
- *Množina agentů A*. Agenti jsou podmnožinou objektů a jsou schopné vykonávat určité akce. Představují aktivní entity systému.
- *Množina polí F*. Agenti mohou šířit do svého okolí prostřednictvím prostředí *E* signály v podobě polí a jiná pole mohou naopak vnímat.

- *Množina míst L* určujících možné polohy objektů (z množiny *O*) v prostoru, tj. v prostředí *E*.
- *Množina relací R*, které vzájemně spojují objekty a také agenty.
- *Množina operací Op*, dávajících agentům schopnost vnímat, manipulovat, vytvářet, odstraňovat objekty z *O*, a reprezentujících především akce agentů.
- *Množina operátorů U*, reprezentujících aplikace operací z *Op* na prostředí a reakcí světa na tento pokus o jeho modifikaci. Operátory *U* jsou nazývány *zákony univerza*. Takto lze definovat obecný multiagentový systém, který může být i prostorový. [3]

Multiagentový hierarchický systém můžeme demonstrovat na ilustračním příklade (Obr. 6), kde A_0 (např. agent dopravní sítě) rozděluje správu celé sítě S_0 na dvě částečné správy regionů S_1 a S_2 . Hovoříme, že agent A_0 *delimituje* správu sítě na dva rovnocenné „regionální“ agenty A_1 a A_2, přičemž od nich může být informovaný o závažných skutečnostech, tykající se celé sítě a též jim může sám odevzdávat pro ně důležité regionální informace.[1]



Obr. 6: Hierarchická struktura agentů a modelů (2)

Předpokládá se, že v každém ze dvou regionů se budou vykonávat stejné správní činnosti. Každý z regionálních agentů požívá na plnění svých cílů dalších úzce specifikovaných agentů, na kterých *deleguje* část svých kompetencí. Z uvedeného obrázku je zřejmé, že regionální agent A_1 využívá pro splnění daných úloh jinou strukturu podřadných agentů než jeho „kolega“ A_2. Z jiného pohledu je možné na hierarchickou strukturu agentů (množinu *A*) pohlížet jako na hierarchickou strukturu

modelů (množinu M, $|M|=|A|$), přičemž každý model $M_i \in M$, ($M_i \subseteq A$) sestavenou ze stromu agentů s kořenem/agentem $A_i \in A$, $i=0,...,|A|-1$. Agenta A_i nazýváme představitelem (*bossem*) modelu M_i, přičemž M_i alternativně označujeme jako $^{model}A_i$..[1]

Modely na nejnižší úrovní hierarchie jsou jedno-agentové modely. V algebraickém vyjádření můžeme potom strukturu modelů M_0 z obrázku vyjádřit výrazem $M_0[M_1(M_3, M_4, M_5), M_2(M_6, M_7(M_8, M_9))]$, přičemž modely v hranatých závorkách vznikli delimitací a modely v kulatých závorkách vznikly delegováním funkcí jejich nadřízeného modelu.[1]

2.8.1 Komunikace

Akce, které agent vykonává transformují stav prostředí. V multiagentní systému však tvoří agentovo okolí i ostatní agenti, kteří se nachází v tomto systému. Z hlediska agenta je proto každý agent pouze dalším prvkem systému vůči kterému může působit – vykonávat akce. Podobně jako v případě neagentních prvků okolí, může být agentovým záměrem změnit vnitřní (mentální) stav jiného agenta. Obecně se rozlišují dva přístupy, kterými může agent mentální stav jiného agenta změnit.

- Nepřímé ovlivnění: Agent nepůsobí přímo na jiného agenta, ale mění stav jeho okolí tak, aby ten při kontaktu s tímto okolím změnil svůj postoj žádaným směrem.
- Přímé ovlivnění: Agent působí na jiného agenta přímo a jediným možným způsobem tohoto ovlivnění je komunikace.

Důvodů ke komunikaci může mít agent několik. Já zde uvádím šest základních typů dialogu.

- Dotazování: Agent hledá nějakou informaci a dotazuje se jiného agenta, o kterém věří, že mu tuto informaci může poskytnout.
- Hledání informace: Agenti společně pátrají po informaci, která není známá žádnému z nich.
- Přesvědčování: Agent se snaží přesvědčit jiného agenta, aby přijal některé z jeho záměrů.
- Vyjednávání: Agenti vyjednávají o podmínkách sdílení prostředků, nebo o poskytnutí služeb tak, aby všichni zúčastnění dosáhli maximálního zisku.

- Porada: Cílem porady je nalézt řešení problému, které je v zájmu všech zúčastněných. Agenti poskytují své znalosti a schopnosti a usnáší se na dalším postupu.
- Eristický dialog: Eristický dialog je dialog, během kterého si agenti vyměňují expresivně informace za účelem dosažení svých záměrů. Tyto informace nejsou ani logickou podporou argumentu, ani vyjednáváním či dotazováním. Typickým takovým dialogem je hádka.

Vlastní komunikace je proces, během kterého si dva nebo více agentů vyměňují informace ve formě elementárních komunikačních zpráv. Každá elementární komunikační zpráva má odesílatele, příjemce, obsah a informaci o typu, který určuje význam obsahu zprávy.

Pokud agent chce komunikovat, musí mít schopnost nalézt partnera vhodného pro záměry, kterých hodlá touto komunikací dosáhnout. Řešení nastíněného problému může vycházet z nástrojů, které jsou již nyní využívány pro distribuované systémy, jako například vysílání (broadcasting) nebo nástěnky. Jiné přístupy nabízí výsledky bádání přímo v oblasti agentních systémů. Hledání vhodného partnera pro komunikaci může být řešeno například přes adresáře služeb, nebo adresáře agentů, přítomných na agentních platformách. Dále lze využít speciálních agentů pro zprostředkovávání komunikace nebo pro vyhledávání služeb, ti se nazývají mediátoři, brokeři, nebo facilitátoři. Posledním známým přístupem jsou sociální modely. Komunikace v sociálních modelech je založena na vysílání požadavku uvnitř skupiny a teprve v případě neúspěchu se požadavek vysílá i k ostatním skupinám v systému.

Posledním tématem agentní komunikace jsou interakční protokoly. Tyto protokoly udávají pravidla komunikace mezi agenty. Komunikační protokoly se liší podle druhu komunikace a částečně kopírují rozdělení typů dialogu uvedené výše.

Každý komunikační jazyk pro komunikaci agentů je, stejně jako všechny ostatní jazyky, dán abecedou, syntaxí a sémantikou.

V současné době je hlavním jazykem pro komunikaci agentů ACL (Agent Communication Language). ACL vychází z jazyka KQML (Knowledge Query Manipulation Language), řeší některé nedostatky, kvůli kterým býval KQML kritizován a jeho standardizace je uvedená ve specifikacích FIPA.

2.8.2 Nástroje pro tvorbu MAS

Rád bych zde uvedl ve stručnosti některé programy pro tvorbu Multiagentových systému.

Prvním nástrojem je *OLAN* a je vyvinutý v programovacím jazyku Java. Vývojové prostředí se skládá z modelovací části, konfigurační části a části sloužící ke generování zdrojových kódů. Modelovací část slouží k vytvoření základní struktury modelu. Komponenty mohou být jednoduché, skládající se jen z implementační části nebo složitější, kde se definují i různá propojení. Do určité míry je zde možné vytvářet vlastní typy komponent simulačního modelu. Nástroj vychází z výhod jazyka Java. Konfigurační nástroj vytváří novou strukturu, která definuje propojení mezi jednotlivými komponenty. Nástroj k vygenerování kódu potom vygeneruje soubor ve formátu XML nazývaný vývojový skript a druhý soubor jež obsahuje parametry, které si může uživatel nastavit pro běh simulace. Nástroj potřebuje pro svoje fungování Java runtime prostředí. Jednou z jeho nevýhod je úprava zdrojových kódů uživatelem. Nástroj nedokáže změny zakomponovat do grafické reprezentace modelu a to představuje problém z hlediska kompatibility.

Nástroj *CORMAS* je vybudován na programu *VisualWork*, který byl postaven v prostřednictvím programovacího jazyka *SmallTalk*. Tento nástroj se používá k modelování přírodních a sociálních dynamických systémů. Je založen na principu distribuce a akce. Agent komunikuje s jinými agenty a přebírá stimuly ze svého prostředí. Tento nástroj umí vytvářet entity na rozdíl od nástroje *OLAN*. Nástroj obsahuje grafické prostředí, ve kterém se vizuálně prezentuje vytvářený model. Toto grafické prostředí se používá k vytvoření topologie modelu. Nástroj neobsahuje generátor kódů a výsledky simulace jsou přímo uvnitř nástroje

Nástroj *JACK* byl také vyvinutý v programovacím jazyku Java. Model vytvořený v tomto nástroji obsahuje objektově orientované prostředí. Nástroj se skládá z editoru agentového jazyka, kompilátoru agentového jazyka a simulačního jádra. Agentový jazyk v nástroji *JACK* je jeho vlastní programovací jazyk, kterým je možné pospat model agentů. Kompilátor přepisuje soubory z agentového jazyka do jazyka Java. Jádro vytváří spustitelné prostředí pro programy vytvořené v agentovém

jazyce. Nástroj *JACK* se specializuje na vytváření distribuovaných agentových systémů. Největší nevýhodou je to, že neobsahuje grafické prostředí, které by zpřehlednilo strukturu modelu.

SimCreator je simulační a modelovací nástroj vyvinutý, v jazyku C++. SimCreator je komerční produkt. Nástroj se skládá opět ze tří částí. Je to grafické prostředí, kde uživatel navrhuje model. Dále to jsou kompilátor, který překládá uživatelův model a interpret tento model spouští. Grafické prostředí je jednoduché, ale přehledné a umožňuje uživateli správu komponent a jejich vztahů. Nástroj umožňuje i správu modulů vytvořených v minulosti. Před spuštěním si může uživatel verifikovat model. Verifikace se týká jen těch komponent, které byly přidané od poslední kompilace nebo ke komponentám změněným. Nevýhodou je to, že uživatel nemá možnost zasáhnout přímo do zdrojového kódu, což může býti omezující v konfiguraci modelu. Dalším problémem je způsob komunikace mezi agenty. Komunikace probíhá pomocí číselných konstant místo zpráv. Se zvýšenou složitostí modelu stoupá jeho nepřehlednost. Poslední problém spočívá v tom, že vystavěný model není samospustitelný, ale dá se spustit pouze v tomto nástroji.

Model Design Interface je založený na jazyku MAML a byl vytvořen v Maďarsku. MAML je založen na volně dostupných knihovnách Swarm, které tvoří standartní simulační balík. Výhodu knihoven je to, že obsahují generátory náhodných čísel a podporu analýzy statistických dat. Po vygenerovaní kódu v jazyce C++ spolu s knihovnami Swarm proběhne kompilace pomocí překladače gcc. Zdrojový kód však může být přeložený a spustitelný i na jiných platformách. Největší nevýhodou je opět nedostatečná flexibilita při vytváření složitých vazeb. V nástroji je obtížné definovat hierarchické vazby s prvky delimitace a delegováním pravomocí na podřízených agentech.

ZEUS je nástroj pro vytváření multiagentních systémů. Zahrnuje možnosti reprezentace znalostí, plánování, komunikace a sociálních vazeb. Je implementován v jazyku Java a jeho silnou stránkou je vizualizace návrhu agentního systému. Multiagentní systém je vytvářen hierarchicky ve třech vrstvách – definiční, sociální a organizační. Na definiční úrovni se definují agenti, jejich znalosti, cíle, prostředky a schopnosti práce v systému. V sociální vrstvě se určuje postavení agentů v sociální

skupině, zahrnující znalosti o jiných agentech v této skupině a informace o jejich dostupnosti. Organizační vrstva stanovuje způsob komunikace a vyjednávání, strategie rozdělování úloh, apod.

Existují i jiné nástroje (např. Jade, Jam...), ale buď jsou vyhrazené pouze na specifickou oblast, nebo jsou příliš obecné. Největší nevýhodou stávajících nástrojů je to, že jsou málo flexibilní pro větší počet agentů v modelu. Dále poskytují jen omezenou sadu komponent a uživatel nemá potřebnou volnost při tvorbě komponent nových či modifikaci stávajících. Při větším počtu agentů v modelu tyto nástroje nemají dostatečný standart při vytváření hierarchických vazeb. Ve velkých modelech je také značně komplikovaná vnitřní logika agentů a nástroje by měli tuto logiku vhodně modelovat.

3 Architektury MAS

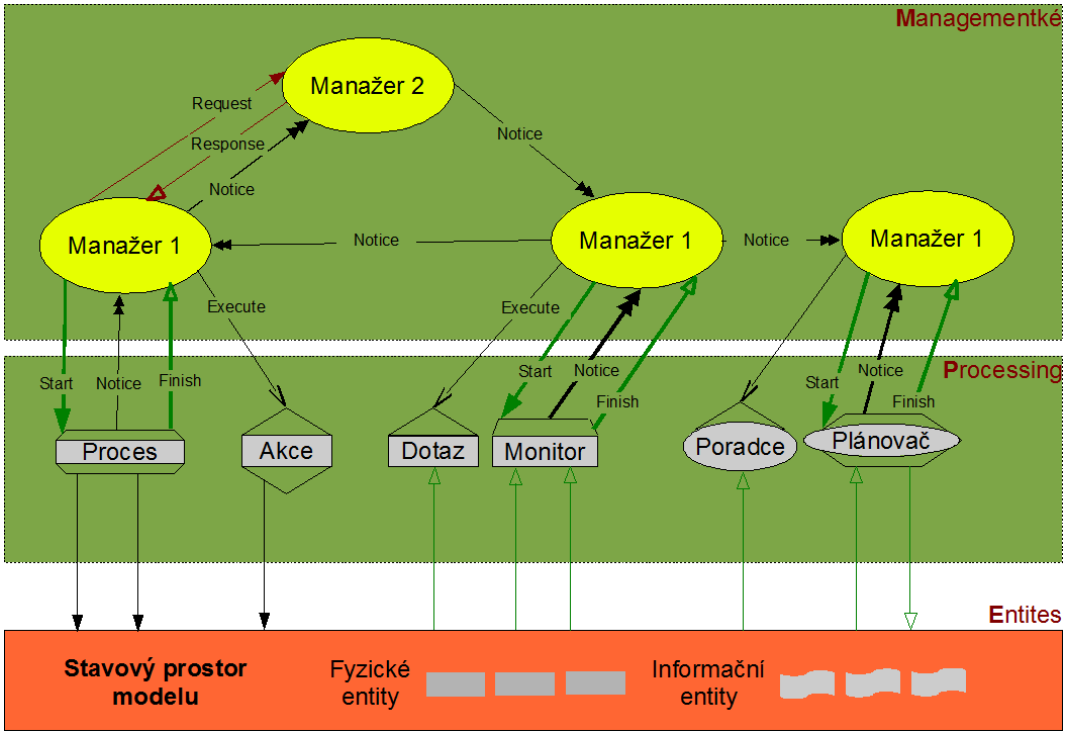
Agentově orientovaná simulace má v současnosti pevné místo při zkoumání komplexních obslužných systémů. Některé architektury jsou ve fázi konceptu, jiné jsou přesně přizpůsobené určité aplikaci a neumožňují tak využití v modelech z jiné oblasti. Existuje však několik architektur jež si kladou za cíl být všeobecně použitelné při tvorbě komplexních simulačních modelů. Nejvíce se budu věnovat architektuře *ABAsim*.

3.1 Architektura ABASim

ABASim (**A**gent-**B**ased **A**rchitecture of **sim**ulation model) architektura byla vyvinutá kolektivem pracovníků na Fakultě řízení a informatiky v Žilinské univerzitě. Tato architektura umožňuje tvořit flexibilní simulační modely komplexních obslužných systémů. Simulační model je složen z kooperujících reaktivních agentů, kteří jsou organizování v hierarchické struktuře. Hierarchická struktura je typická pro obslužné systémy. Každý agent této architektury se skládá z komponent, které zabezpečují *senzorickou*, *výkonnou* a *komunikační* činnost agenta. Z této architektury vychází i moje práce a tedy i program, který jsem vytvořil. Následující text se bude podrobě věnovat této architektuře.

V ABAsim architektuře jsou systémy modelované z hlediska *rozhodování* a *řízení*. Jak bylo uvedeno výše řídící prvky mají určité kompetence a jsou hierarchicky organizované. Řídící prvek má zabezpečit předem definovaný cíl. Při plnění

cíle se rozhoduje zda bude spolupracovat se svými podřízenými řídícími prvky nebo zda danou úlohu zpracuje sám. Řídící prvky se modelují pomocí agentů. Agent tvoří základní stavební a funkční jednotku v modelu obsahuje totiž funkce potřebné pro řízení a aplikaci svých rozhodnutí. Každý model se skládá z jednoho nebo více kooperujících agentů. Agent má několik hlavních funkcí jako je *identifikace cíle*, *rozpoznává situaci*, *rozhoduje o řešení* a *vykonává řešení*. Agent při rozhodování o řešení může použít funkce *návrhu řešení problému* a *kooperaci* s jinými agenty nebo s člověkem.



Obr. 7: Vrstvy simulačního modelu (2)

Rozklad agenta na jeho jednotlivé vnitřní součásti do dvou skupin (manažeři a asistenti) nastiňuje myšlenku dívat se na celý model jako na systém založený z jednotlivých vrstev (Obr. 7). Jednak je to *vrstva managementu*, která se rozhoduje a vydává pokyny pro realizaci svých rozhodnutí. Potom je zde *vrstva zpracování* kde její součásti asistují managementu a realizují požadované výkony ve vymezeném prostoru simulačního modelu.

V modelu je možné uvažovat také o další vrstvě. *Vrstva entit* vytváří stavový prostor simulačního modelu a nese *informační* a *fyzické* entity. Fyzické entity odpovídají všem prvkům simulovaného modelu. Všechny fyzické entity jsou modelovány

komplexní modelovou strukturou, která odráží stav simulovaného modelu. *Informační* entity nesou informace, které nejsou přímo součástí fyzických entit, ale jsou potřebné k řízení systému nebo popisují chování modelu v čase.

V popisované architektuře existuje komunikace jednak mezi agenty a jednak vnitřní komunikace mezi managerem a jeho asistenty. Oba dva druhy jsou v ABAsim architektuře realizované pomocí zasílání zpráv a z tohoto hlediska je možné označit ABAsim architekturu jako zprávově orientovanou. Existují zde tři vrstvy, na kterých probíhá komunikace. Následuje přehled jaké zprávy se používají v jaké vrstvě.

Pro komunikaci mezi agenty (mezi jejich managery) se používají typy zpráv:

- *Notice* (oznámení), je zpráva, na kterou se neočekává žádná odpověď.
- *Request* (žádost) je zpráva obsahující určitý požadavek (na poskytnutí služby či informace), přičemž odesílatel očekává na ni od adresáta odpověď.
- *Response* (reakce/odezva), která reprezentuje povinnou odpověď na zprávu typu *Request*, přičemž může být doručená pouze jejímu odesílateli.[1]

Pokyny, které může manager dávat svým asistentům jsou realizované pomocí následujících typů zpráv:

- *Start*, po přijetí zprávy tohoto druhu začne adresát (musí to být kontinuální asistent) svojí autonomní činnost.
- *Break*, představuje jediný způsob, jak může manažer ovlivnit autonomní běh kontinuálního asistenta, který je po přijetí této zprávy nezvratně ukončen (tento typ se používá jen ve výjimečně a to v případech, kdy nastaly v simulačním modelu takové podmínky, při kterých již nemá smysl, aby kontinuální asistent dokončil svoji činnost).
- *Execute* je specifickým typem zprávy, určené pro promptního asistenta, který okamžitě vykoná odpovídající činnost a obratem zprávu vrátí svému manageru (odesílateli) s vyznačeným výsledkem svoji činnosti.[1]

Posledním druhem zpráv, které se v architektuře ABAsim používají jsou zprávy, které odesílají kontinuální asistenti v čase svojí autonomní činnosti:

- *Finish* je zpráva, kterou každý z kontinuálních asistentů povinně zašle svému manažerovi v okamžiku své činnosti.

- *Notice* jsou zprávy, které mohou být zaslané v zásadě kdykoliv v průběhu činnosti kontinuálního asistenta, kdy kontinuální asistent potřebuje oznámit svému managerů výskyt pro něho důležité situace.
- *Hold* je jedinou zprávou, která obsahuje tzv. *časovou pečeť* určující čas na její dokončení, přičemž tuto zprávu si kontinuální asistent zásadně posílá sám sobě (kontinuální asistent může po odeslání této zprávy opět pokračovat ve svojí činnosti až od toho okamžiku simulačního času, kdy je mu tato zpráva doručena − do té doby je neaktivní). [1]

V ABAsim architektuře se běh simulačního času zabezpečuje výlučně zasláním zpráv typu *Hold*, a tedy časová synchronizace modelu se vlastně realizuje synchronizací činností kontinuálních asistentů. Ve vrstvě managementu se zprávy s časovými pečetěmi nepoužívají, a tedy zde se simulační čas neovládá a nesynchronizuje. V případě aplikace architektury v distribuovaném prostředí je nutné vybavit i zprávy managementu časovými pečetěmi pro případ, že jsou tyto zprávy zasílané do jiného uzlu v síti s jiným lokálním simulačním časem než má odesílatel.

Po vysvětlení zásad komunikace v ABAsim architektuře je zřejmé, že popsaná komunikace slouží hlavně pro realizaci diskrétních simulačních modelů. Nicméně je možné rozšířit mechanizmus komunikace tak, aby se dal použít i při tvorbě diskrétně-spojitých simulačních modelů.

3.2 Architektura Cougaar

Tato architektura je výsledkem výzkumu americké armády v oblasti nasazení agentových systému na modelování logistického systému armády. Architektura je určena pro distribuční simulaci s důrazem na modularitu a škálovatelnost. Agenti jsou v této architektuře složeni z komponent, každý agent je komplexní a různorodý vzhledem k ostatním agentům, pracuje autonomně a asynchronně. Agenti se sdružují do komunit a z komunit se tvoří společenstva agentů. Společenství agentů je hierarchickou organizační strukturou. Jeden agent může být součástí i více komunit.

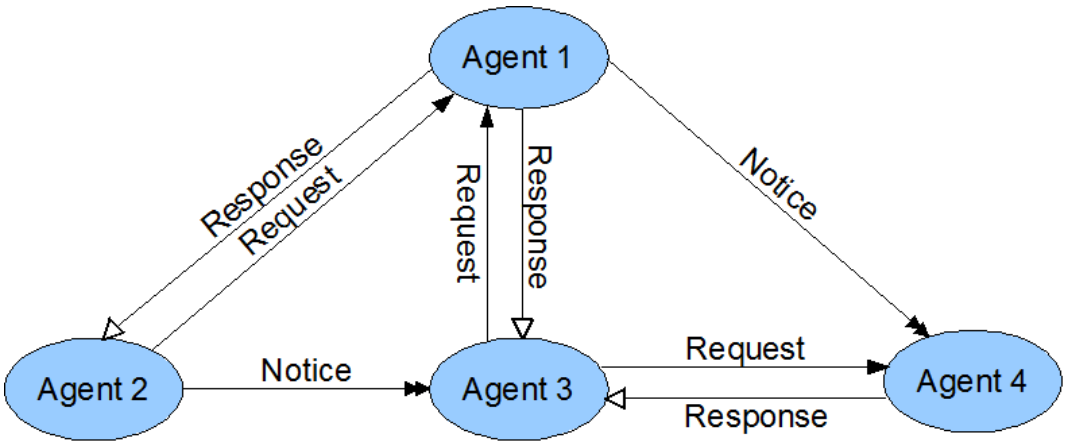
3.3 Architektura Cybele

Architektura *Cybele* (*OpenCybele*) je určená pro vývoj distribuovaných agentových aplikací. Není omezena pouze na simulační modely, ale lze v ní tvořit i diskrétní a spojitou simulaci. Tuto architekturu můžeme aplikovat na simulační modely

logistických systémů. Agenti této architektury jsou autonomní a podporují mobilitu. Jako v předchozí architektuře i zde jsou agenti organizováni do společenstev. Komunikace je řešena v podobě zasílaní informací o změnách atributů, o které agent projevil zájem. V současné době existuje odlehčená volně dostupná verze *OpenCybele.*

4 Stávající vizualizace v ABAsim architektuře

Zde bych rád opět upozornil na to, že tato práce se zabývá pouze vizualizací vrstvy managementu v architektuře ABAsim. Stávající zobrazení je výhodné a přehledné pouze pro počet agentů nepřesahující deset. Po překročení tohoto počtu agentů se stává model značně nepřehledným a orientace v něm je značně komplikovaná, zvláště má-li každý agent komunikovat s každým agentem. Navíc jeli u, každé zprávy se musí uvedeno o jakou zprávu se jedná a také popis zprávy (např. notice: zákazník odešel). Následuje stručný přehled výhod a nevýhod a na uvedeném obrázku (Obr. 8) je ilustrovaná stávající forma vizualizace komunikace mezi čtyrmi agenty.



Obr. 8: Stávající komunikace

Výhody

- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující několik agentů, kteří nekomunikují pomocí zpráv Request, Response a Notice každý s každým.

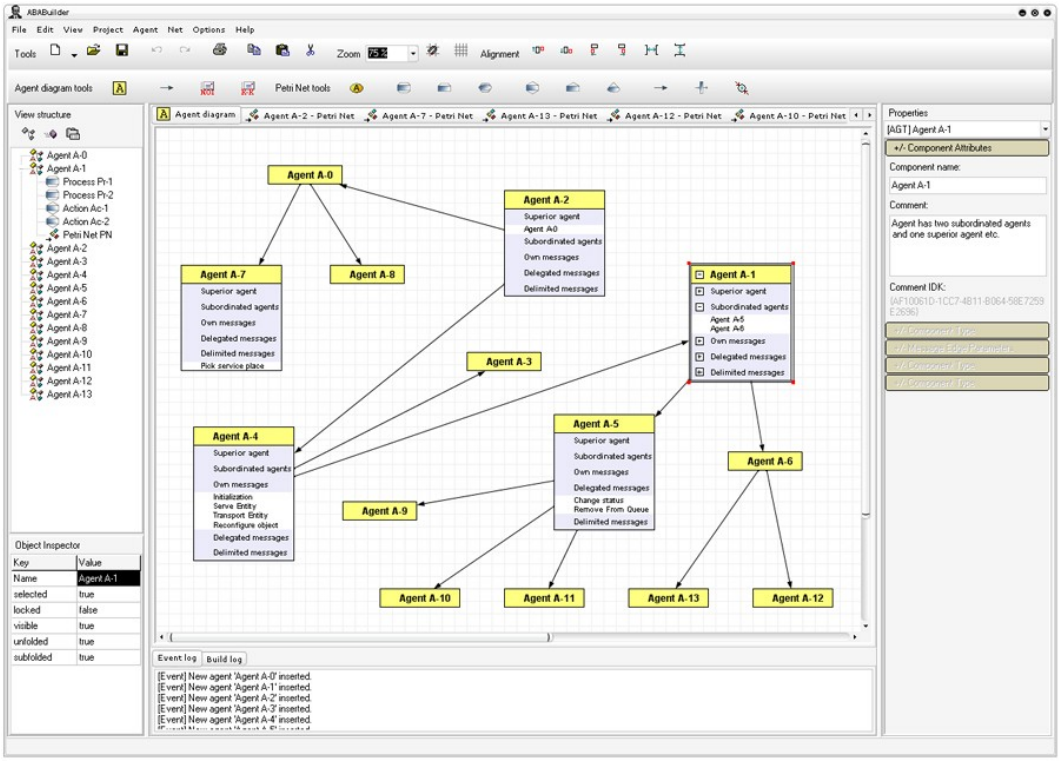
Nevýhody

- Nepřehledné při použití více jak deseti agentů, kteří komunikují (rozesílají zprávy) každý s každým.

4.1 ABABUILDER

V této době již existuje nástroj, který vychází z architektury ABAsim a nazývá se *ABAbuilder*. Tento program byl vyvinut na fakultě řízení a informatiky v Žilině. Program se skládá ze čtyř modulů je to modul *vizuálního prostředí*, *modul verifikace*, *generátor zdrojových kódů* a *analyzátor zdrojových kódů*.

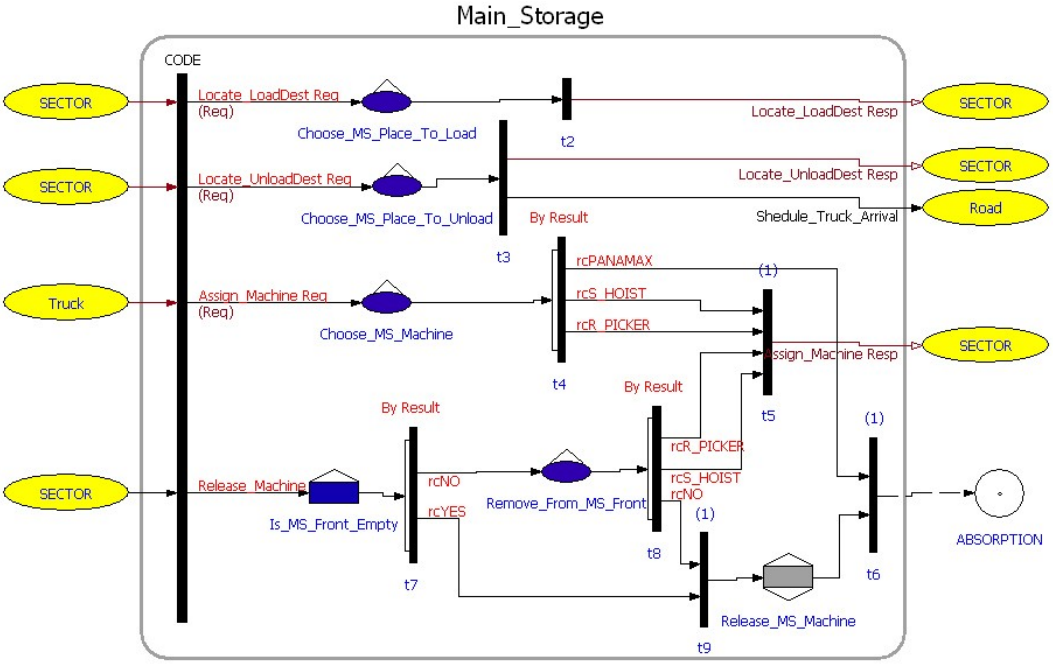
- V modulu *vizuálního prostředí* uživatel nastavuje komunikaci agentů mezi sebou prostřednictvím zpráv *Request*, *Response* a *Notice*. Agent je základním stavebním prvkem. Výhodou tohoto prostředí je aplikace filtrů, kdy je možné pracovat pouze s jednou specifikovanou skupinou agentů.
- Modul *verifikace* kontroluje změny ve vytvářeném modelu a poskytuje uživateli dva druhy hlášení. Jednak je to hlášení o vzniklých chybách a potom je to varování o nedokončeném stavu.
- *Generátor zdrojového kódu* potřebuje znát konceptuální model a až po skončení fáze návrhu modelu může kód vygenerovat. Pro každého agenta se vygeneruje vlastní jednotka (unita) a pro každého asistenta třída nesoucí jméno asistenta.



Obr. 9: ABAbuilder agentový diagram (3)

- Modul pro *analýzu zdrojových kódů* analyzuje kódy u vytvořených programových rutin. Verifikuje návratové hodnoty a exekuční metody asistentů modelu. Pokud je nutné změnit základní strukturu modelu pak by docházelo ke střetu těch částí kódů, které byli doplněné. Tento modul se stará o jejich správné umístění.

Simulační jádro programu je založené na struktuře vytvořené v ABAbuilderu přímo pro účely simulace. Jádro je součástí simulačního modelu a skládá se z několika částí. Při samotném modelování je v programu na výběr ze dvou možností jak model sestavovat. První možností je *agentový diagram*, který popisuje hierarchickou vazbu mezi agenty a jejich komunikační mechanizmus. Druhá možnost je Petriho síť konkrétního agenta, která popisuje vnitřní fungování-logiku agenta. Na následujících obrázcích je ukázka jednak agentového diagramu v ABAbuilderu tak i možnost reprezentace pomocí *Petriho sítě*. Na obrázku 9 je ukázka z programu je to návrh modelu pomocí agentového diagramu a na obrázku 10 je ukázka Petriho sítě vytvořené v ABAbuilderu.



Obr. 10: Petriho síť navrhnutá v ABAbuilderu (3)

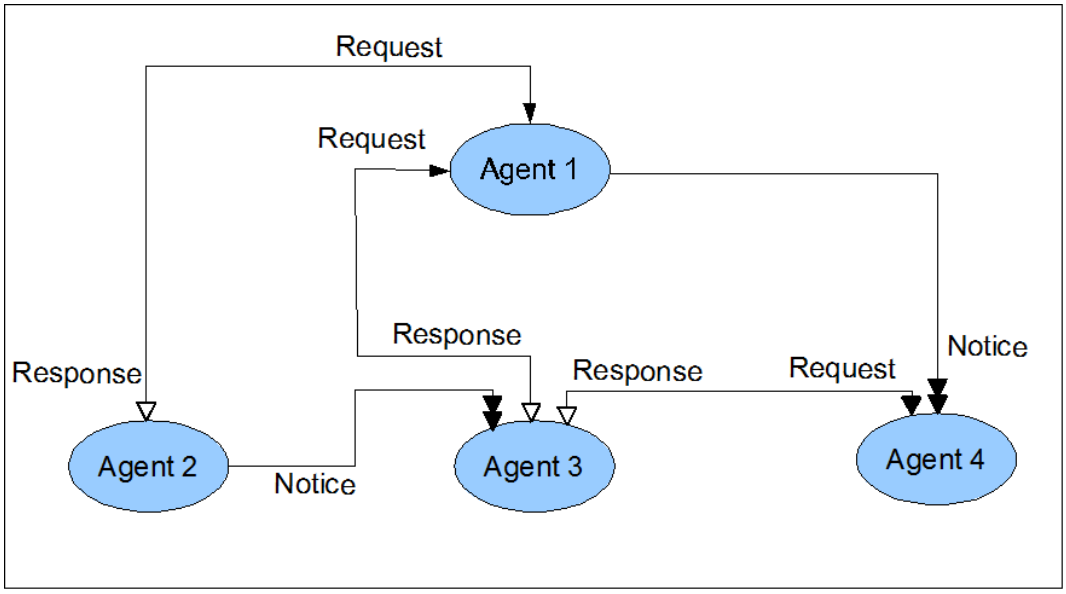
5 Navrhovaná vizualizace v ABAsim architektuře

Navrhovaná vizualizace je rozdělena do dvou skupin, podle toho k čemu a jak se použije. Já jsem zvolil dvě varianty. První varianta jsou verze určené pro tisk na papír. Druhá je varianta je určena pro elektronické formy prezentace, které nabízí lepší a přehlednější možnosti prezentace. V budoucnu by mohl existovat nějaký proces, který byl byl schopen převádět realizaci ze jedné formy na druhou.

5.1 Verze pro tisk na papír

5.1.1 Vizualizace s rovnoběžnými hranami

Toto zobrazení (Obr. 11) vychází z klasického zobrazení ovšem upravuje jej. Vychází z toho, že každá zpráva Request musí být v páru s Response, proto je Request zakončen značkou pro Response. Návrh se tak stává jednodušším a přehlednějším.



Obr. 11: Vizualizace s rovnoběžnými hranami

Výhody

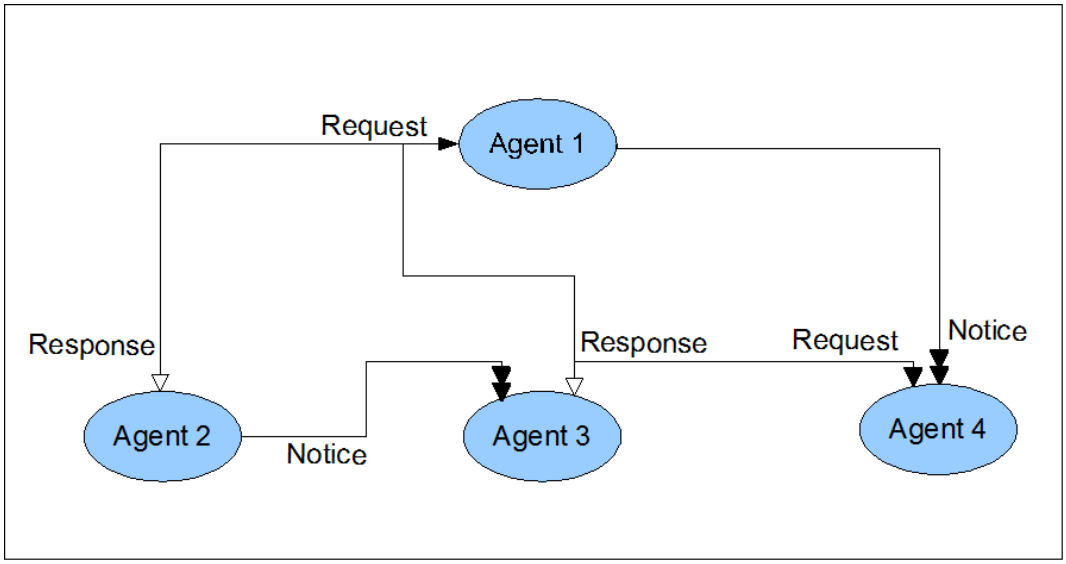
- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehlednější již pro větší počet agentů.

Nevýhody

- Nepřehledné při použití více agentů, kteří komunikují každý s každým.

5.1.2 Vizualizace s rovnoběžnými redukovanými hranami

Zobrazení dvě (Obr. 12) vychází z předchozího návrhu zobrazení a dále jej zjednodušuje v tom, že pro každého agenta uvažuje pouze jeden vstup pro zprávy Response, Request a Notice a jeden výstup pro totéž. V předchozích možnostech se uvažovalo o tom že pro každou vazbu bude existovat vždy jedna spojnice mezi agenty. Tento návrh vzchází z toho, že je možné spojit všechny stejné vazby do jedné, tedy např. místo deseti Requestu vstupujících do jednoho agenta bude jen jeden reprezentující všech deset. Dále tato možnost počítá opět s tím, že Response a Request jsou zprávy párové. Agent má dále pouze jeden výstup a tím je notice. Notice je vedena z jednoho místa na agentovy do libovolného počtu agentů.



Obr. 12: Vizualizace s rovnoběžnými redukovanými hranami

Výhody

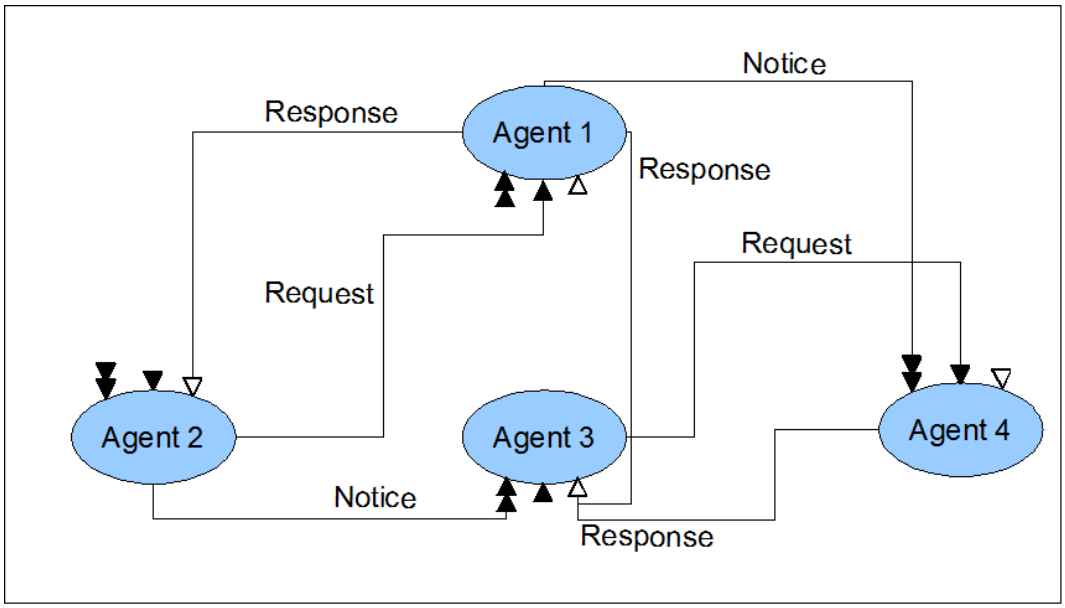
- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.

Nevýhody

- Při větším počtu agentů špatná orientace ve vazbách mezi zprávami zasílané jednotlivými agenty.

5.1.3 Vizualizace s pevně def. vstupy zpráv

Třetí možnost je založena na principu vstupu a výstupu. Každý agent má v jedné své části (nahoře,dole,vpravo,vlevo) vstupy na druhé nejlépe opačné výstupy. Vstupy jsou jako v předchozí variantě pouze tři a to pro Response, Request a Notice a jsou sjednoceny, to znamená pokud agent dostává deset zpráv typu Notice od různých agentů pak se všechny sbíhají v jednu (více obrázek 13). U agenta jsou vstupy zavedeny, i když je agent momentálně nepoužívá. Výstupy jsou naopak vedeny buď z protilehlé strany agenta nebo jak uživatel zvolí, ale počet výstupů může být značný a každý výstup je reprezentován čárou.



Obr. 13: Vizualizace s pevně def. vstupy zpráv

Výhody

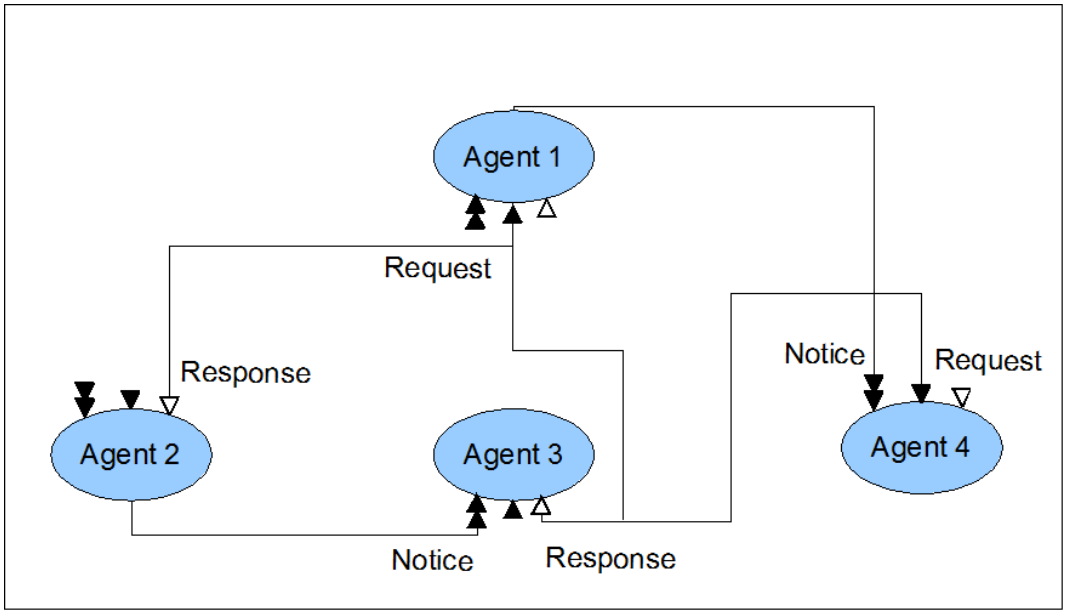
- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.

Nevýhody

- Při větším počtu agentů špatná dohledatelnost vazeb mezi agenty.

5.1.4 Vizualizace s redukovanými pevně def. stupy zpráv

Tato možnost opět vychází z toho, že zprávy Request-Response jsou párové. Je to kombinace předchozích dvou možnosti. Jedná se o možnost velice přehlednou i když v rozsáhlých projektech jsou zprávy těžko dohledatelné. Více na obrázku 14.



Obr. 14: Vizualizace s redukovanými pevně def. stupy zpráv

Výhody

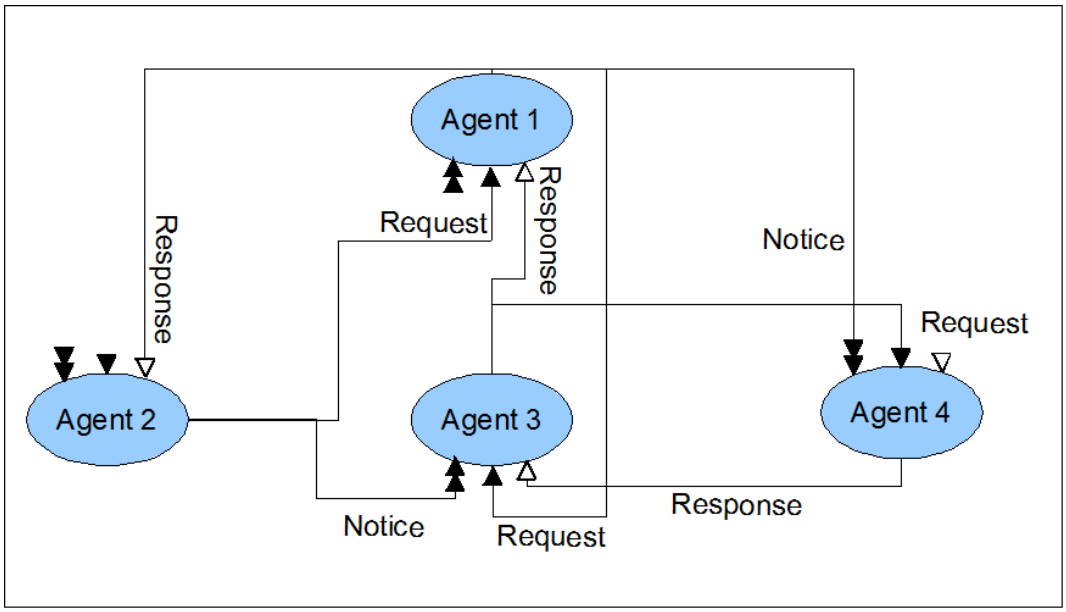
- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít request, response a notice vazby každý s každým.

Nevýhody

- Pokud je model rozsáhlí pak je obtížnější najít patřičnou komunikaci mezi agenty.

5.1.5 Vizualizace s rovn. hranami a pevně def. vstupy

Je to modifikovaná předchozí možnost s tím, že výstupy vycházejí z jednoho místa na agentovy. Vstupy jsou zachovány jako v předchozím případě. Výhody a nevýhody jsou srovnatelné jako v předchozí verzi. Rozdíl mezi těmito verzemi je lépe patrný z následujícího obrázku.



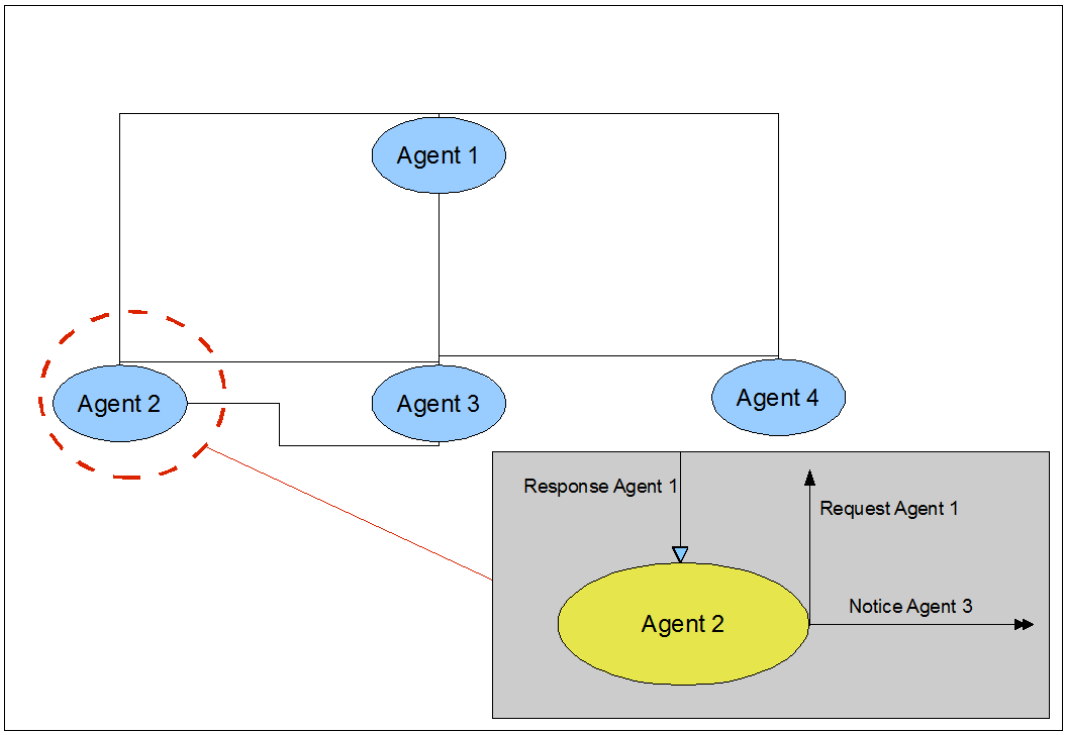
Obr. 15: Vizualizace s rovn. hranami a pevně def. vstupy

5.2 Verze pro elektronickou formu

5.2.1 Kombinovaná verze s náhledem

V tomto zobrazení jsou zprávy agentů reprezentovány pouze čárami, které jako takové ztrácí vypovídací schopnost o druhu zprávy (Obr. 16). Nicméně tyto čáry nám ukazují s kým agent komunikuje. Jednotlivé druhy zpráv se nám ukáží po zazoomování, či při kliku na agenta. Po zvětšení-přiblížení k agentovi je již jasné s kým agent komunikuje neboť vazby jsou již vidět v plném rozsahu.

Při kreslení na papír je tato varianta značně nepraktická, neboť by obnášela nejprve nakreslit celý systém a poté znovu překreslovat jednotlivé agenty a jejich vazby na ostatní agenty v systému. Zobrazuje se pouze to s kým agent komunikuje.



Obr. 16: Kombinovaná verze s náhledem

Výhody

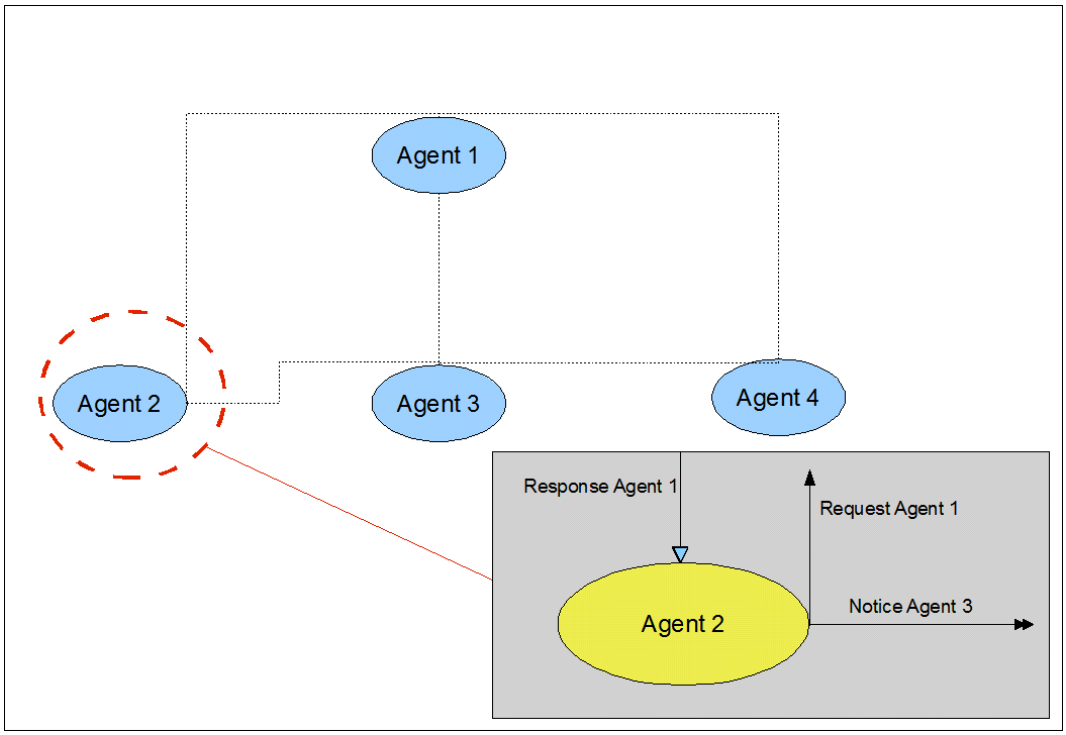
- Možné použití při kreslení jak v elektronické podobě tak na papír (i když na papíře je dosti špatně zpracovatelná).
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.

Nevýhody

- Každá zpráva je zde zastoupena pouze čarou, z toho vyplývá, že je nemožné na první pohled odhalit o jakou zprávu se jedná. Také se stoupajícím počtem agentů roste nepřehlednost.

5.2.2 Redukovaná kombinovaná verze s náhledem

Toto zobrazení je variantou předešlého zobrazení. Tato varianta se odlišuje tím, že zde již není reprezentována každá vazba, ale stačí pokud existuje alespoň jedna vazba mezi agenty. Při zvětšení či kliku je pak možné vidět jaké zprávy kam agent vysílá a od jakého agenta případně zprávy dostává (Obr. 17). Zobrazuje se pouze to s kým agent komunikuje.



Obr. 17: Redukovaná kombinovaná verze s náhledem

Výhody

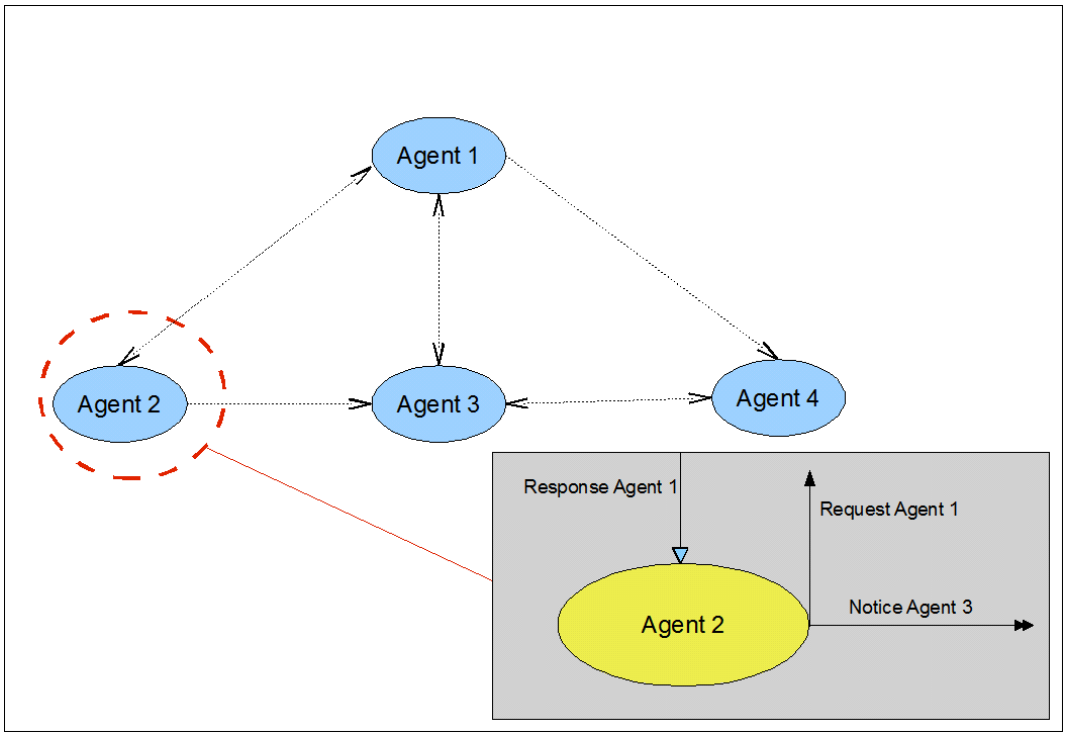
- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů. Nevadí pokud by každý agent zasílal všechny tři druhy zpráv (Request, Response a Notice) všem zbývajícím agentům.

Nevýhody

- Horší realizace při přenosu na papírovou formu.

5.2.3 Verze s náhledem a orientací

Tato varianta ještě rozšiřuje předchozí variantu tímto způsobem: Pokud agent zasílá zprávy jiným agentům je charakter této zprávy znázorněn např.: pokud vede od 1. agenta zpráva Notice k 4. agentu pak se zobrazí šipka u agenta 4. Pokud 1. agent posílá zprávu Request 3. agentu pak se zobrazí šipka u obou. Výhodou této varianty zobrazení je toto: pokud agent vysílá zprávu Request a Notice zároveň zobrazí se vazba mezi agenty jen jako obousměrná. Ta má větší prioritu a až při zoomu či kliknutí se objeví detaily a jednotlivé druhy zpráv, které agent vysílá či přijímá (Obr. 18). Zobrazuje se pouze to s kým agent komunikuje.



Obr. 18: Verze s náhledem a orientací

Výhody

- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.

Nevýhody

- Špatná realizace kresleni na papír (nejprve je nutné nakreslit celý systém a poté ješte jednotlivé agenty)zejména pří větším počtu agentů.

5.2.4 Verze s orientací a okolím agenta

Poslední variantou modifikace klasického zobrazení je možnost kombinace předchozí varianty s variantou 5.2.2. Tato varianta by měla být nejpřehlednější. Její výhoda je v tom, že při zvětšení náhledu na vybraného agenta zobrazuje Request-Response zprávu jako jednu zprávu. Zobrazuje se pouze to s kým agent komunikuje.

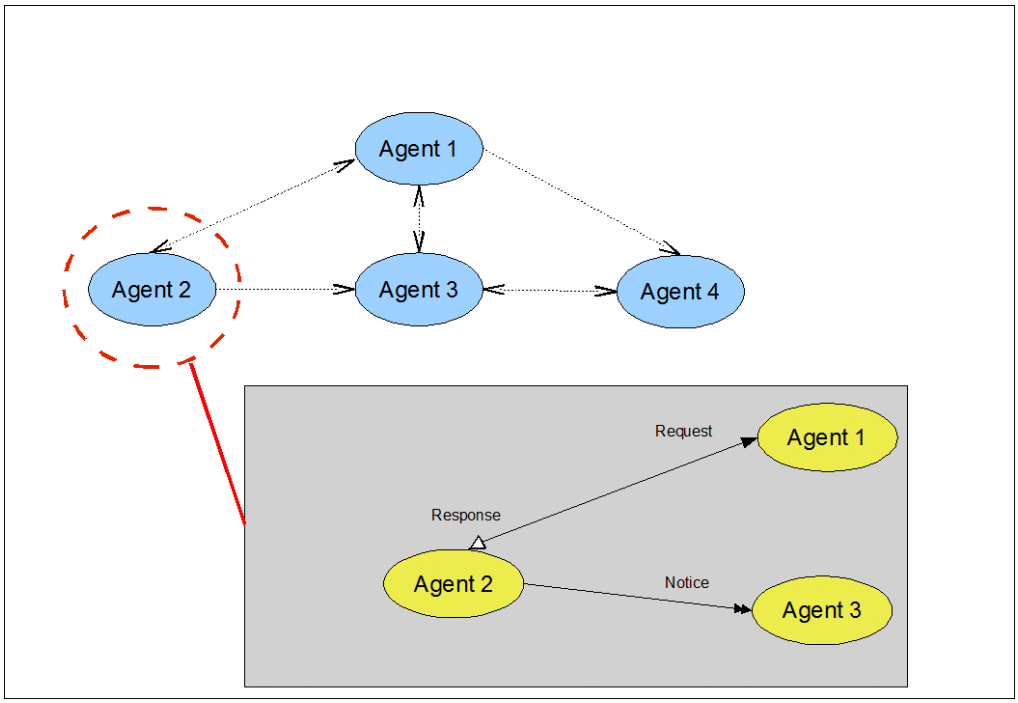
Oproti předchozí variantě je při zvětšení zřejmé to s kým daný agent komunikuje a jaké druhy zpáv přijímá, případně odesílá. Při kliku na agenta či zazoomování je tedy jen určitý výsek z celého systému (Obr. 19). V této variantě je asi nejhorší její použití při kreslení na papír.

Výhody

- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.

Nevýhody

- Špatná realizace kresleni na papír (nejprve je nutné nakreslit celý systém a poté ještě jednotlivé agenty)zejména pří větším počtu agentů.



Obr. 19: Verze s orientací a okolím agenta

5.2.5 Verze se zprávami v těle agenta

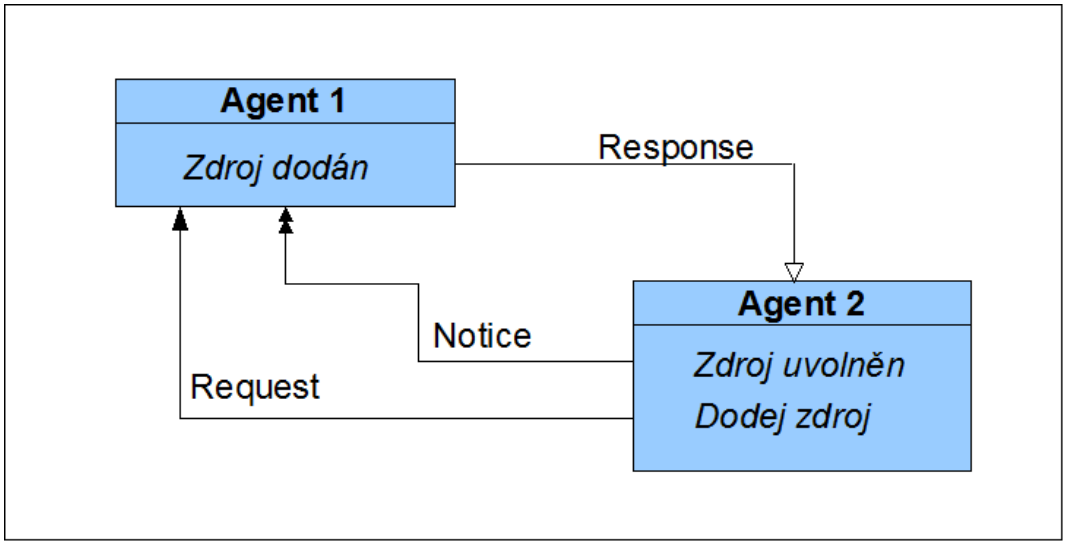
Doposud jednotlivé varianty více či méně vycházely ze základního zobrazení, pokud tedy agent komunikoval s jiným, muselo se u vazby mezi nimi uvést o jaký druh zpráv jde a případně o popis. Tato varianta vychází z toho, že vlastní zprávy agenta jsou vedeny v jeho těle, to poskytuje zjednodušení v tom, pokud agent zasílá tutéž zprávu více agentům. Dále v této variantě se nepoužívá zobrazení párové vazby Request-Response. Agenti samotní jsou reprezentování čtverci i obdélníky, to záleží na počtu zpráv, které zasílají do modelu ostatním agentům.

Výhody

- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.

Nevýhody

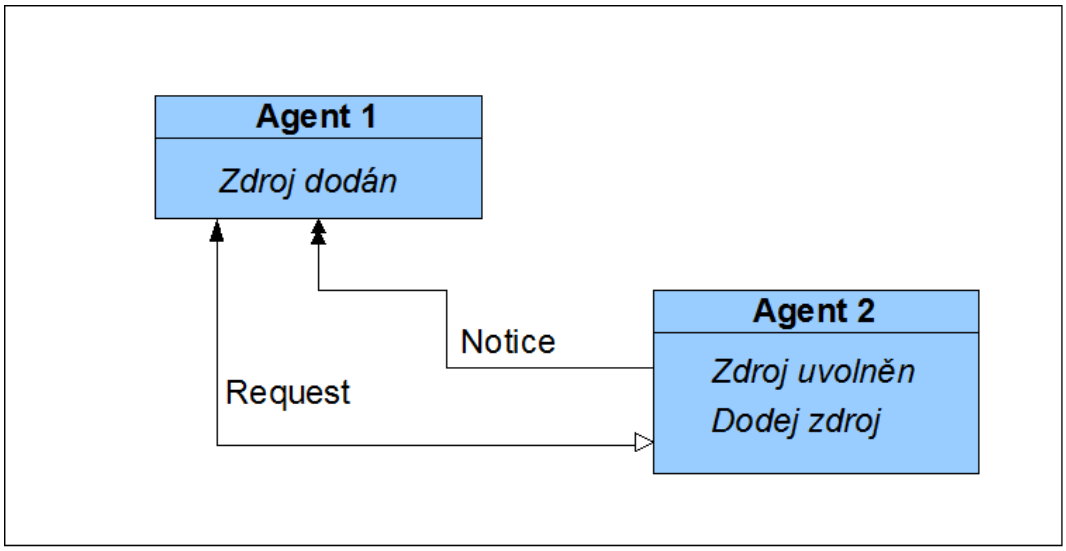
- Horší realizace kresleni na papír zejména pří větším počtu agentů.



Obr. 20: Verze s zprávami v těle agenta

5.2.6 Redukovaná verze se zprávami v těle agenta

Jedná se o rozšířenou nadstavbu předchozí možnosti. Zjednodušení, které je v této variantě vychází z toho, že vazba Request-Response je párová, resp. na každý Request se odpoví Response proto v této variantě



Obr. 21: Redukovaná verze s zprávami v těle agenta

Výhody

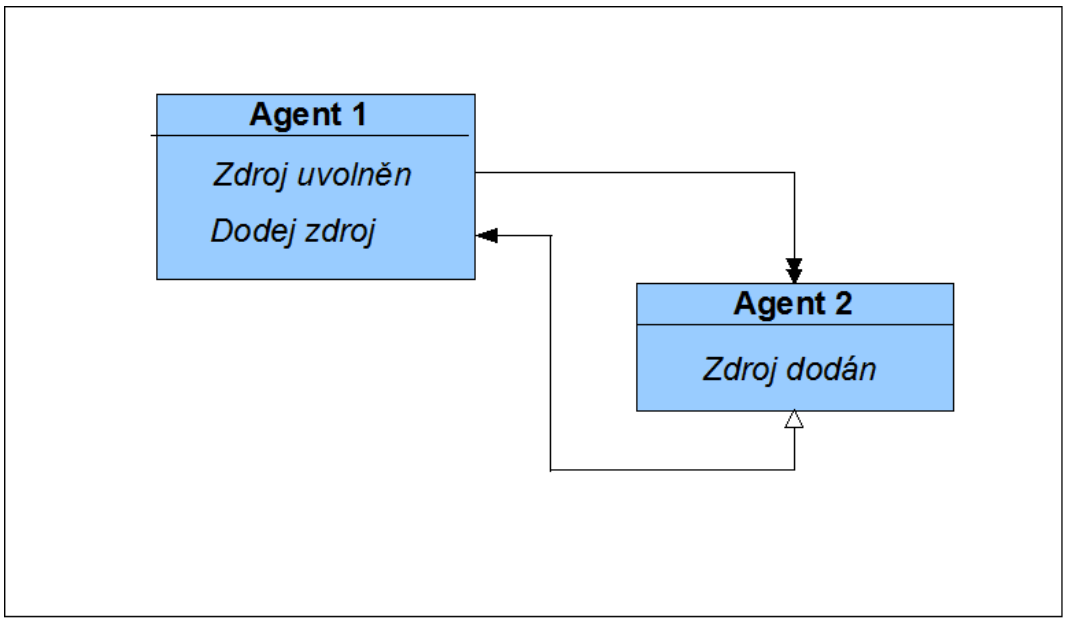
- Možné použití při kreslení jak v elektronické podobě tak na papír.
- Přehledné pro systém obsahující již větší počet agentů, kteří mohou mít Request, Response a Notice vazby každý s každým.
- Ještě přehlednější jak předchozí varianta

Nevýhody

- Horší realizace kresleni na papír zejména pří větším počtu agentů.

5.2.7 Verze s opačnou adresací zpráv

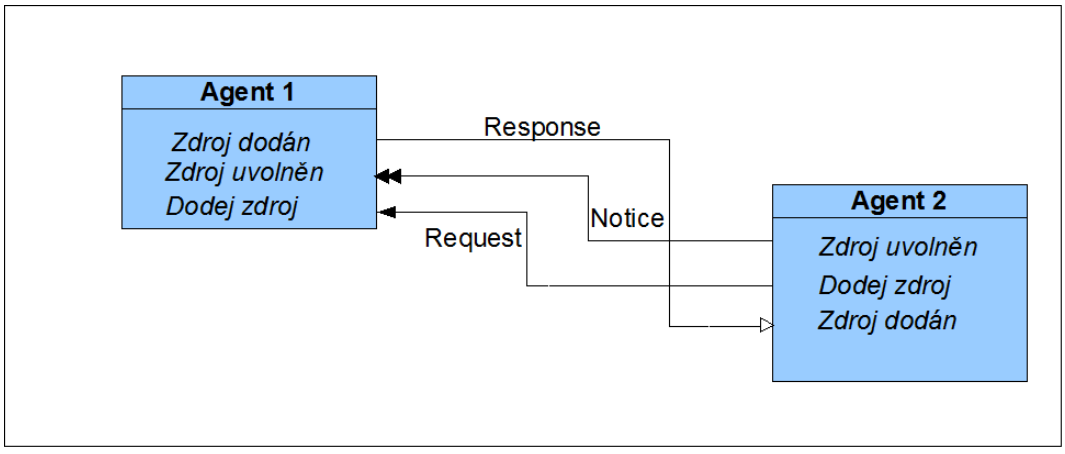
Předešlé varianty zobrazení vycházejí z toho, že agent obdrží zprávu určitého typu s určitým obsahem. Na předchozích obrázcích jsou uvedeny příklady, kdy agent 2 obdrží zprávu Notice s kódem „Přišel zákazník“. Tato varianta vychází z předpokladu toho, že kódy zpráv budou zakomponovány v adresovaných agentech (více na obrázku). Tato možnost má stejné výhody i nevýhody jako možnost předchozí.



Obr. 22: Verze s opačnou adresací zpráv

5.2.8 Verze se stejnými kódy

Tato možnost je založena na tom, že jak v adresátovi tak v odesílateli je uložen stejný kód aby bylo jasné, o který kód se jedná a adresát tento kód uměl zpracovat. Je pak možné, že v těle agentů bude spousta kódů a budou tak vznikat „mohutní agenti“.



Obr. 23: Verze se stejnými kódy

6 Podpora vizualizace s knihovnou NetBeans

6.1 NetBeans Platform

NetBeans je open source platforma jazyka Java. V současnosti je vývoj NetBeans podporován firmou Sun. Při vzniku Netbeans bylo hlavním cílem vytvořit integrované vývojové prostředí pro vývoj Java aplikací. V současné době NetBeans

platforma obsahuje nejen balíky produktů pro tvorbu a vývoj desktop aplikací, ale také obsahuje balíky pro tvorbu aplikací servrových či aplikací určených pro mobilní zařízení. Kterékoliv prvky této platformy mohou být použity ve vlastním produktu postaveném na této platformě. Ve výsledném řešení tedy mohou být použity již někým otestované a použité části.

NetBeans jsou šiřitelné pod CDDL (Common Development and Distribution License) licencí, která umožňuje využití jak pro nekomerční, tak i komerční účely. Tato licence umožňuje vytvářet a distribuovat moduly a aplikace založené na NetBeans a používat součásti NetBeans ve vlastních aplikacích.

Důvodem pro výběr NetBeans RCP (Ritch Client Programming) platformy je používaní Swing komponent. Jádro je postaveno z dat v prostředí Oracle. Podobně i prostředí Eclipse dovoluje použít Swing komponenty, ale oproti NetBeans není vzhled tak konzistentní. Aplikace vytvořené v NetBeans prostředí mohou používat dynamicky nové moduly bez nutnosti celou aplikaci znovu instalovat pro přidání nového modulu. Vývoj aplikace se skládá z implementace NetBeans modulů, které mohou běžet v rámci NetBeans IDE nebo v rámci NetBeans Platform. Existuje velké množství modulů pro NetBeans, které mohou být využity ve vyvíjené aplikaci.

Vlastnosti NetBeans platformy:

- Správa uživatelského rozhraní – platforma nabízí okna, menu, nástrojové lišty a jiné komponenty pro prezentaci. Vývojář píše akce, které systém bude nabízet, a soustředí se tak na vývoj aplikační logiky.
- Prezentace dat – platforma nabízí bohatou škálu nástrojů pro zobrazení a editaci dat.
- Správa nastavení – ukládání a správa nastavení aplikace je bezpečná, jednoduchá a prováděna automaticky.
- Grafická editace – Drag and Drop editace dat apod.
- Editor textu – velmi silný editor textu vycházející z NetBeans IDE editoru, který může být upraven a rozšířen pro editaci různých typů souborů.
- Framework pro průvodce – nástroj pro vytváření rozšiřitelných průvodců, které vedou uživatele v průběhu definování komplexních úkolů.

- Široký sortiment dalších komponent – využití komponent pro verzování, specializované editory, vzdálený přístup ke zdrojům apod.
- Systém aktualizací – aktualizace aplikací jsou založeny na internetových technologiích a udržují systém aktuální.

Vývoj v prostředí NetBeans Platform se provádí pomocí průvodců v IDE a zjednodušuje tak psaní konfiguračních souborů. Jak bylo již zmíněno výše, neexistuje v současné době dokumentace, která by obsáhla platformu jako celek. V následujících měsících by ale měla být publikována kniha Rich Client Programming: Plugging into the NetBeans™ Platform, která by podle prvních ohlasů měla být oficiální dokumentací k platformě.[4]

6.1.1 Open API a jádro platformy

Samotná platforma se skládá z jádra a množiny rozhraní Open API, které se nacházejí v balíčku org.netbeans.core.* resp. org.openide.*. Jádro implementuje řadu hlavních rozhraní Open API potřebných pro samotný běh aplikace. Veškeré části výsledné aplikace se implementují do modulů. NetBeans modul je Java archív obsahující implementaci, která používá Open API, a manifest soubor, který obsahuje identifikaci, verzi implementace a závislosti na ostatních modulech.[4]

6.1.2 System FileSystem

Jedná se o virtuální systém souborů, který slouží jako úložiště konfiguračních údajů aplikace. Moduly využívají tento prostor pro definici vlastních přípojných bodů. Modul vytvoří svůj virtuální adresář a zdokumentuje, jaké virtuální soubory se mohou v tomto adresáři nacházet. Typicky se do těchto adresářů vkládají instance souborů. Ostatní moduly, aniž by cokoli věděly o implementaci svého okolí, mohou do těchto adresářů vkládat záznamy. Jedná se o volnou vazbu na základě implementace rozhraní definovaných ve veřejných API vrstvách.[4]

Od tohoto okamžiku se modul pro správu akcí postará o vytvoření instance třídy StartMiningAction, a vytvoření příslušných tlačítek, položek menu, nástrojové lišty apod. Tento prostor můžou moduly modifikovat prostřednictvím layer.xml souborů. Záznamy se spojují do jedné konfigurace přístupné všem modulům. Tento přístup úzce souvisí s tím, že moduly mají vlastní zavaděč tříd (class loader), a nemají přístup k třídám (class path) ostatních modulů, i přesto, že běží v rámci jednoho vir-

tuálního stroje Javy. Instance tříd vytváří System FileSystem. Tím je zajištěno volné vázání (tzv. loose coupling) a programování do rozhraní, místo do konkrétních tříd. System FileSystem je základním kamenem při implementaci modulů.[4]

6.1.3 FileObject, DataObject a Node

Platforma dokáže rozpoznávat různé druhy souborů, které mají v aplikaci vlastní ikony, položky v menu a chování. Soubory jsou reprezentovány jako FileObjects, což je obalová třída kolem java.io.File třídy. To, co je v aplikaci zobrazeno, jsou instance třídy Node, které poskytují operace a prezentaci různých objektů, mezi které se řadí i soubory. Mezi Nodes a FileObjects se zařazují DataObjects.[4]

DataObject slouží jako vyšší abstrakce nad souborem. Rozumí obsahu daného druhu souboru. To znamená, že pro každý druh souboru existuje příslušný DataObject.[4]

6.2 Knihovna NetBeans Visual Library

NetBeans Visual Library (NVL) je knihovnou, která je použitelná pro obecnou vizualizaci včetně podpory zobrazení grafů. Je základem pro veškeré grafické editory NetBeans, jako je vizuální editace konfiguračních souborů (Struts, JSF apod.) nebo definování procesů (BPEL).

Samotné programování je podobné jako v Javě Swing. Základem je plocha, na kterou se kreslí. Tato plocha se nazývá v NVL *SCENE*. Tato scéna si potom veškeré komponenty zanesené na ni ukládá do stromové struktury. Existuje více druhů scén, výběr nejvhodnější scény závisí na tom, co v dané aplikaci chceme vizualizovat. Základní je Scene, tato obsahuje další poddruhy scén jako je ObjectScene, GraphPinScene<N,E,P> (parametry označují **N**ode, **E**dge a **P**in při konstrukci), VMDGraphScene, GraphScene<N,E> (parametry označují **N**ode a **E**dge při konstrukci), GraphScene.StringGraph. Scéna se většinou vytvoří samostatně, ale potom se na ni „nanese“ nějaký objekt. Většinou se jedná o formulář nebo panel vytvořený ve Swing prostředí. Následuje příklad vytvoření a připojení scény na panel.

```
Scene novaScena = new Scene ();
jPanel1.add(novaScena.createScene().createView());
```

Základním stavebním prvkem je Widget existuje opět mnoho druhů jako je například LabelWidget, ListWidget, IconWidget, ScrollWidget atd. Tyto prvky mají

různé vlastnosti a je možné upravovat jejich atributy jako je poloha na scéně, rámeček kolem nich, výšku, šířku, barvu apod. V následujícím zdrojovém kódu je uvedeno vytvoření LabelWidgetu. Tento LabelWidget se jmenuje *first*, nese text *label 1*, ve scéně se objeví na souřadnicích 100, 100, bude mít kolem sebe ohraničení čarou a bude s ním možné pohybovat.

```
LabelWidget first = new LabelWidget(scene, "label 1");
first.setPreferredLocation(new Point(100,100));
first.setBorder(BorderFactory.createLineBorder());
mainLayer.addChild(first);
first.getActions().addAction(ActionFactory.createMoveAction());
```

Další důležitou komponentou jsou vrstvy tzv. *Layers*. Výhodou vícevrstvého programu je to, že mohou být zpřístupněny nebo ne. To znamená, že i komponenty na nich zobrazené mohou být zpřístupněny nebo stejně jako vrstva neviditelné. Z toho vyplývá, že pokud se vykresluje nějaký složitější objekt nebo skupina objektů pak je možné upravovat je separovaně právě pomocí toho že jednotlivé části objektů nebo objekty budou mít svoji vlastní vrstvu. V následujícím příkladě je uvedeno vytvoření vrstvy s názvem *mainLayer* na scéně (uvedeno výše).

```
LayerWidget mainLayer = new LayerWidget (scene);
scene.addChild(mainLayer);
```

Další součástí je spojení komponent. Spojovat lze všechny Widgety pomocí prvku *Connection*. Pomocí *Connection* se nastavuje jak má propojení vypadat, tedy nejen to jaké objekty se mají navzájem propojit, ale také tvar spojení, zakončení spojení. Pokud jsou prvky spojení mobilní, pak i to jak má spojení vypadat pokud se jeden nebo i více prvku přesune na jiné místo na scéně. V následujícím kódu je uvedeno spojení mezi dvěma widgety (*first* a *second*), spojnice se bude pohybovat po čtvercovém obrysu widgetu, pokud se s nim nějak pohne a bude zakončena plnou šipkou. Spojení se přidá do vrstvy *connectionLayer*.

```
ConnectionWidget connection = new ConnectionWidget (scene);
connection.setSourceAnchor(
          AnchorFactory.createRectangularAnchor(first));
connection.setTargetAnchor(
          AnchorFactory.createRectangularAnchor(second));
connection.setTargetAnchorShape(AnchorShape.TRIANGLE_HOLLOW);
connectionLayer.addChild(connection);
```

Samozřejmě, že toto není kompletní přehled toho co tato knihovna obsahuje, ale je to přehled toho nejdůležitějšího co je využito v připojené aplikaci a zároveň je to základ pro aplikace, které jsou vhodné pro vizualizaci agentů. Mezi další možnosti knihovny patří: Podpora pro kreslení a vytváření grafů, zoom, podpora pro změnu

detailů, animace, editace v místě kliku myší, klávesové zkratky, kombinace kláves a myší pro editaci prvků, různé módy zobrazení, export scény do obrázků nebo do PDF souborů a podpora tisků.

6.3 Aplikace NVL

Existuje již několik aplikací či modulů vytvořených v NVL více či méně vhodných pro vizualizaci agentů. Uvedl bych zde popis některých nejvhodnějších z nich a zároveň ty co byly inspirací pro aplikaci. Velkou výhodou použití NetBeans NVL je zejména v tom, že jednotlivé komponenty mají výbornou dokumentaci a mnohé z nich jsou uvedeny v příkladech, tak že je potom velice snadné je aplikovat.

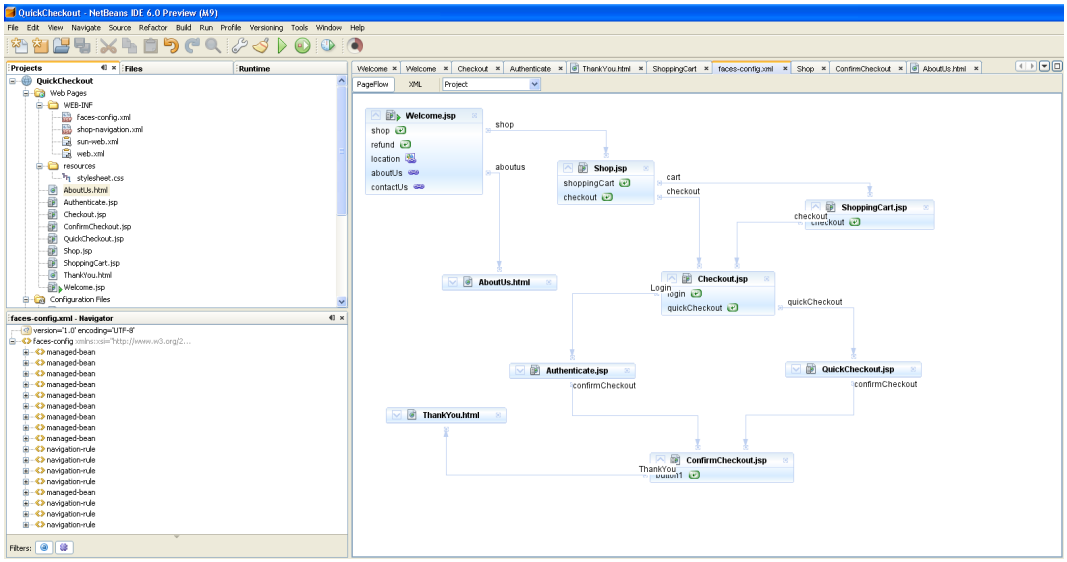
6.3.1 NetBeans Visual Web Project

NestBeans Visual Web umožňuje rychlé budování webových aplikací v prostředí NedBeans pomocí přesouvání, vkládání komponent a datových zdrojů ve *WISIWYG* Visual editoru. Vytváří tak komplexní knihovnu JavaServer a rozmisťuje projekty jako je Web Archív (WAR) do Java Enterprice Edition kontejnerů, mezi které patří Java Systém Aplication Server, JBOSS, BEA WebLogic, IBM WebSphere, Tomkat a jiné. Pomocí Query Editoru se vytvářejí fronty, přidávají se vazby mezi komponenty, specifikují se kriteria a upravuje se zde výsledný vzhled. Pomocí metody „táhni a spusť“ (drag and drop) se vytvářejí vazby na datové služby. V něm obsažený Data Provider umožňuje vazby na jiné heterogenní zdroje dat. Pomocí bodu a kliknutí se definuje uživatelský proud aplikace pomocí standardních JavaServer Face navigace v Page Navigatoru. Vytvoří se konzistentní vzhled pomocí editoru kaskádových stylů (CSS). Pomocí „táhni a spusť“ metody JavaServer Face nastavují vlastnosti komponent a píší se kódy událostí na serveru, pokud uživatel nepreferuje ruční nastavení tagů. Knihovna JavaServer Face obsahuje standardně komponenty, jako jsou tabulky, kalendář, strom, tab. nastavení, nahrávání souborů a mnoho dalších. Na obrázku je ukázka navrhované aplikace.

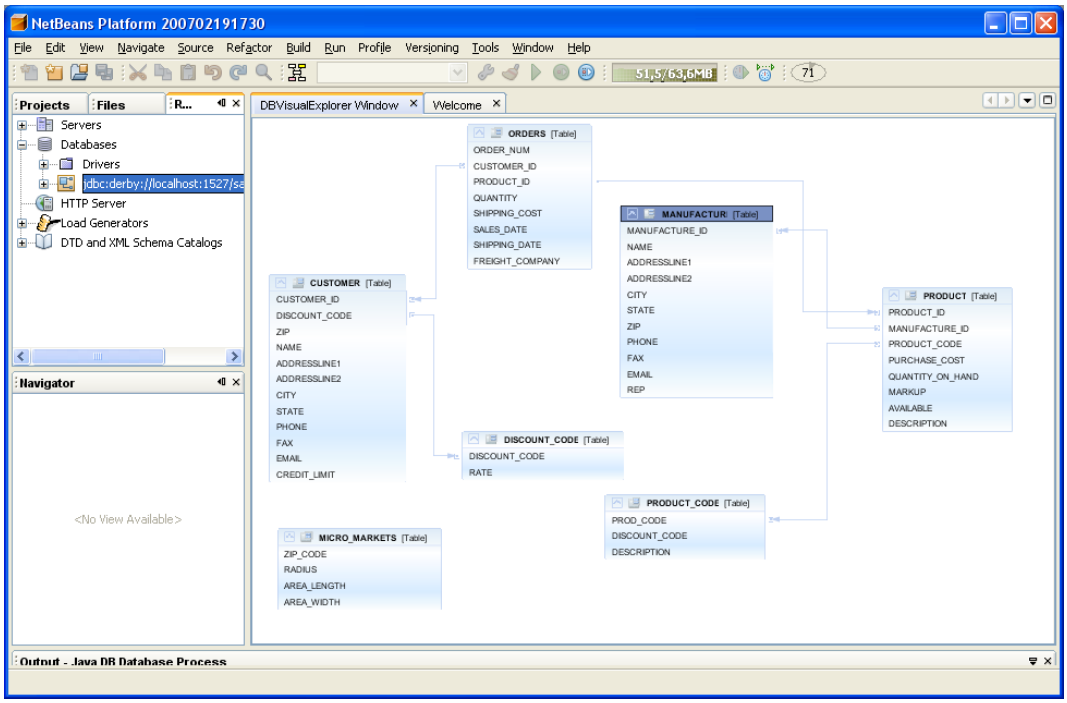
6.3.2 Visual Database Explorer

Toto prostředí je jak název napovídá, určeno pro vizualizaci databáze. Existuje další podobný produkt a to je DbVisualizer, ten má navíc komponentu ukazující strukturu databáze. Hlavní výhodou této vizualizace jsou relace mezi jednotlivými

tabulkami pokud jde o cizí klíče. Vše ostatní je k dispozici ve stromovém zobrazení. Tabulky cizích klíčů mohou být zobrazeny jako graf s tabulkami, které jsou uzly a cizí klíče budou představovat hrany. Obrázek ukazuje možný návrh databáze realizovaný v tomto prostředí.



Obr. 24: Příklad tvorby v Netbeans Visual Web Projektoru (4)



Obr. 25: Návrh databáze v prostředí Visual Databaze Explorer (4)

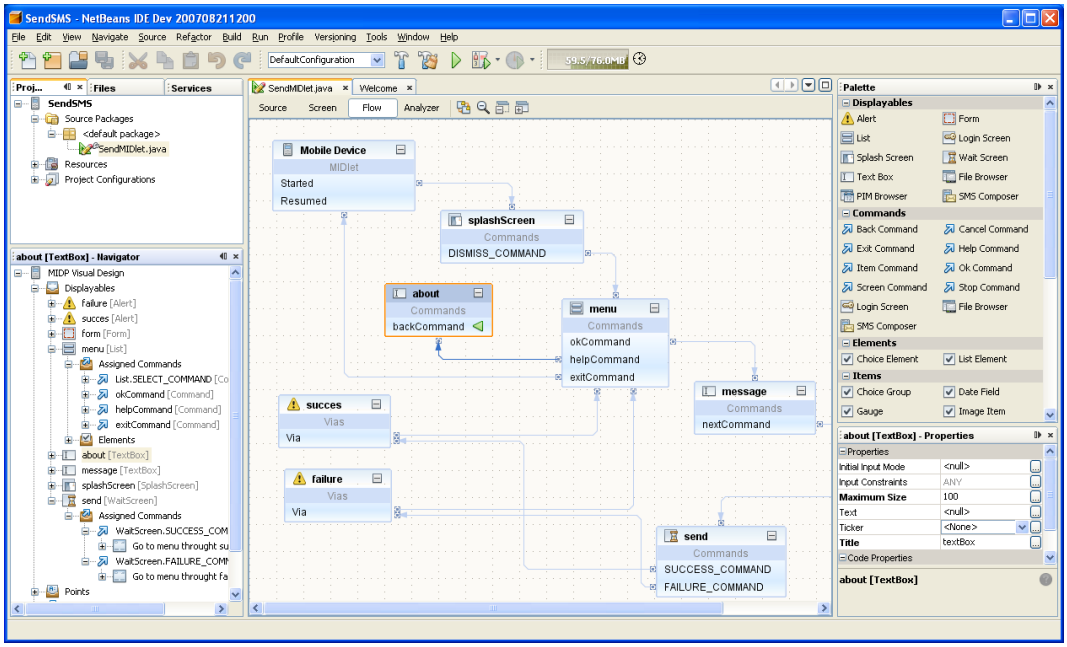
6.3.3 NetBeans Mobility

NetBeans Mobility může být použita pro zápis, test, ladění a nasazování aplikací v Java Micro Edition (Java ME) platformě u mobilních a vestavěných za-

řízení. Podporuje dvě základní konfigurace na platformě Java ME:

- Connected Limited Device Configuration (CLDC) je pro zařízení s menší pamětí a výkonem, než CDC-založené prostředky. Mobile Information Device Profile (MIDP) je založena na CLDC a více než miliarda MIDP zařízení jsou k dispozici po celém světě.
- Konfigurace připojeného zařízení (CDC) je určena pro zařízení s mnohem větší pamětí, výkonností a síťové konektivity, jako jsou chytré telefony, set-top-boxy, internet, zařízení a vestavěné servery.

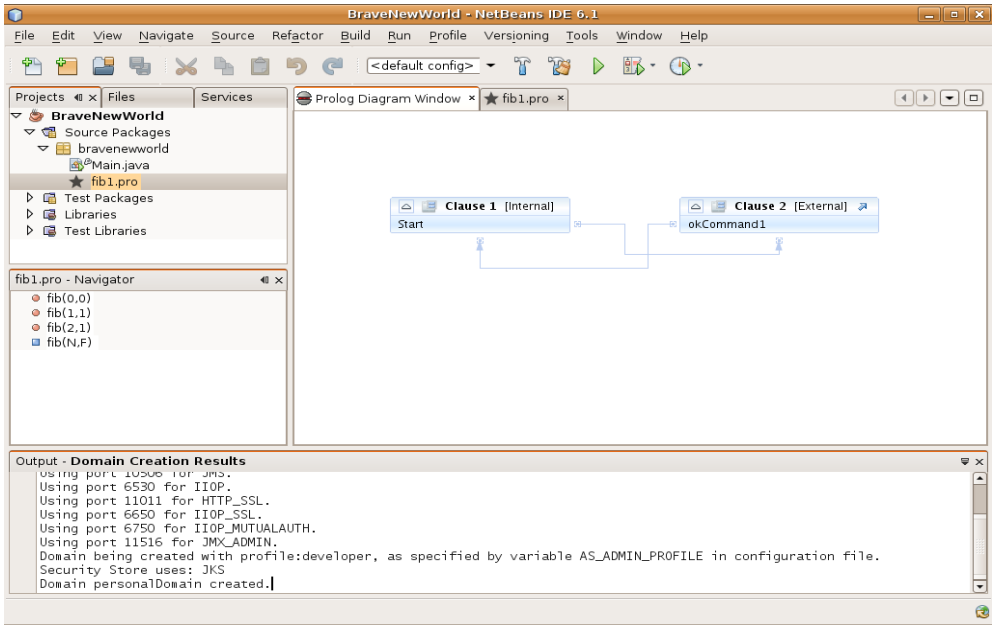
Mobility IDE je obsažena v balíku Sun Java Wireless Toolkit pro CLDC jako výchozí emulátor, ale NetBeans platforma Manager umožňuje snadno integrovat třetí stranu-emulátory od Sony-Ericsson, Nokia a další je možné přizpůsobit nebo rozšířit v testovacím prostředí Java ME aplikacích. Ukázka vývoje testovací aplikace na obrázku 26.



Obr. 26: Aplikace Mobility balíku (4)

6.3.4 Visual Prolog Modeler

Nejedná se o hotovou aplikaci jako v předchozích případech, ale spíše o návod jak takovou aplikaci postavit. Samotný návod a tedy i výsledná aplikace vychází z výše zmíněné aplikace Visual Database Explorer, navíc podporuje zvýraznění syntaxe a v prologu psaný přehledný kód aplikace. Většina zdrojového kódu je popsána a to co není se dá snadno dohledat. Na obrázku je výsledná aplikace.



Obr. 27: Ukázka z programu Visual Data Modeler (4)

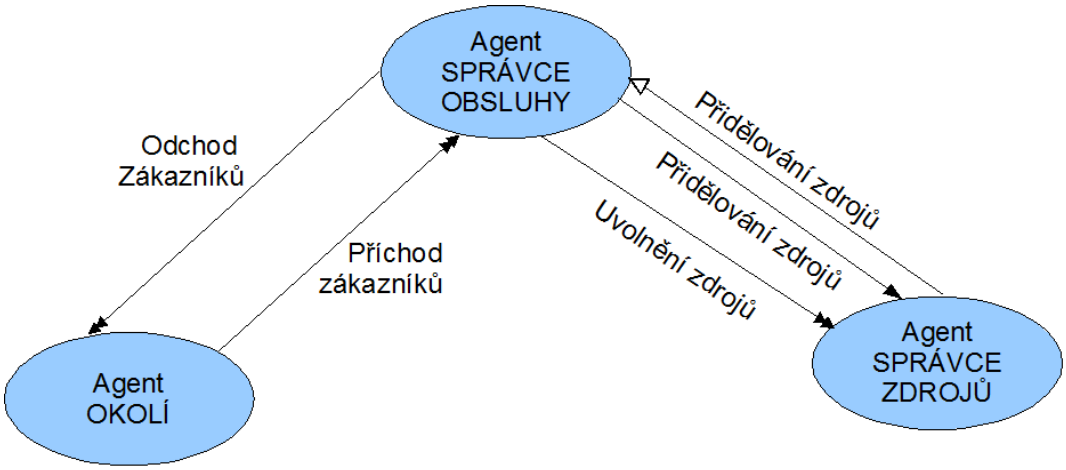
6.4 Příklad a jeho řešení v NVL

V následujícím příkladě budu demonstrovat to, jak se problematika vizualizace agentů řeší v současné době a jaké řešení jsem navrhl a zpracoval jednak pro verzi k tisku a jednak pro uchování v elektronické podobě.

Jedná se o zjednodušený model hromadné obsluhy. Zákazníci vstupují do systému z jeho okolí a absolvují dva typy obsluh a po jejích ukončení systém opouští. První typ obsluhy se váže ke stabilnímu zdroji, ke kterému se zákazník musí přemístit, druhý obslužný zdroj je mobilní (přemístí se k zákazníkovi).

Model uvedeného systému bude tedy obsahovat tři agenty: *Agenta Okolí*, *Agenta Správce obsluhy* a *agenta Správce zdrojů*. Agent Okolí je zodpovědný za kontakt mezi systémem a jeho okolím např. příchody a odchody zákazníků. Druhý agent Správce obsluhy je zodpovědný za organizaci obsluh. Poslední agent má na starosti přidělování a přemisťování zdrojů obsluhy.

Tento model uvedu na obrázku č. 28 v podobě jak by mohl vypadat v prostředí ABAsim ve vrstvě managementu. Z obrázku je potom patrné jak v modelu jeho komponenty mezi sebou komunikují. Agent správce obsluhy žádá v případě potřeby agenta Správce zdrojů o přidělení příslušného zdroje, který je nutné zabezpečit pro vykonání obsluhy zákazníka. Po jeho přidělení a vykonání uvedené obsluhy je příslušný zdroj opět přidělen k dispozici správě obsluhy. Komunikace mezi okolím a správou obsluhy slouží na zabezpečení příchodu zákazníků do a z modelu.

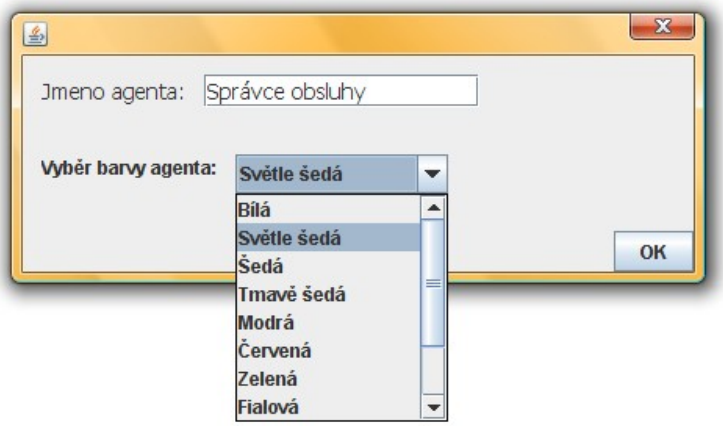


Obr. 28: Agenti obslužného systému

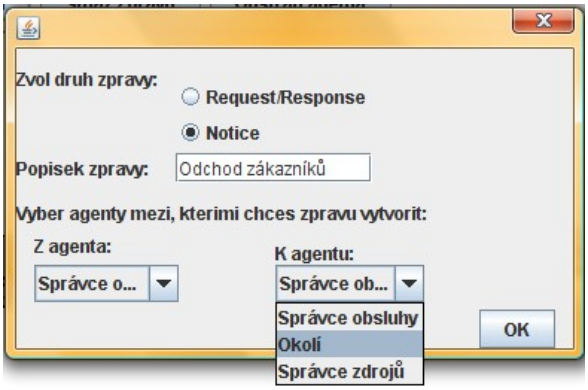
Řešení a tedy i program, který jsem vytvořil vychází z verze uvedené jako „Redukovaná verze s zprávami v těle agenta“ uvedená v kapitole 5.2.6. Tato verze se mi jeví jako nejlepší pro řešení vizualizace pro elektronickou formu a zároveň je tato verze aplikovatelná i pro tisk. Vycházím tedy z toho, že zprávy Request a Response jsou párové a pro jejich reprezentaci postačí pouze jediná spojnice mezi komunikujícími agenty. Program je na obsluhu velice jednoduchý. Bohužel v současné době neexistuje podpora uložení a načtení scény, tedy to co se dá vytvořit pomocí mého programu se nedá uložit. Bohužel neexistuje ani podpora exportu scény do formátu PDF či exportu do obrázku. Podpory, uložení a exportu existují pouze v experimentálním provedení pouze pro určitý typ zobrazení na scéně.

V programu je jednoduché se zorientovat. Pomocí tlačítka „Pridej agenta“ přidáme na scénu agenta. Po stisku tohoto tlačítka se zobrazí dialog s jehož pomocí je možné zadat jméno agenta a barvu pozadí agenta, přičemž předdefinovaná barva pozadí agenta je bílá. Každý agent může mít jinou barvu. Na obrázku 29 je zobrazen tento dialog.

Tlačítko „Pridej zprávu“ se zpřístupní po té, co jsou na scéně min. dva agenti. Po použití tlačítka se otevře dialog pro přidání zprávy mezi agenty. Je možné vybrat si druh zprávy Notice, nebo Request/Response. Dále je zapotřebí zadat text zprávy. Naposled je nutné zadat to, mezi kterými agenty bude komunikace probíhat. Nelze pokračovat pokud je totožný název agenta v obou kolonkách. Na následujícím obrázku 30 je uveden příklad. Notice je reprezentována prázdnou šipkou a Request/Response párová zpráva je reprezentována plnou šipkou.

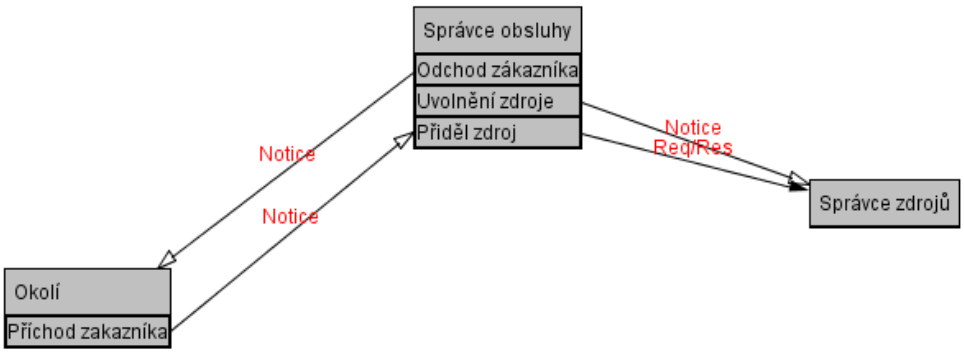


Obr. 29: Dialog pro přidání agenta.



Obr. 30: Dialog přidání zprávy

Výsledná vizualizace modelu uvedeného na obr. 27 by tedy mohla vypadat následovně jak je uvedeno na obr. 31. Nejedná se o jedinou možnou výslednou vizualizaci, která mohla vzniknout v prostředí NetBeans. Aplikace se mohla skládat i z jiných komponent jež jsou modernější, avšak jejich aplikace a následné naprogramování bývá mnohem složitější než je tomu v tomto příkladě, protože některé knihovny podporující tyto prvky jsou v době vzniku této práce stále ve formě experimentu.



Obr. 31: Možná výsledná vizualizace

7 Závěr

Práce je zaměřena na problematiku vizualizace agentů. Shrnuje většinu dostupných variant zobrazení a hodnotí jejich klady a zápory. Tyto varianty jsou pak rozděleny podle toho jak mají být realizované. Buď se jedná o varianty tisku nebo o varianty uchované na elektronickém nosiči dat. Uvedené programy nebo jejich modifikace mohou sloužit k případné vizualizaci. Avšak v současnosti je jen několik málo programů vhodných k tomuto účelu.

Součástí práce je i aplikace, která vychází z poznatků na základě výběru vhodného modelu vizualizace. Aplikace ukazuje nové možnosti při vizualizaci agentů a klade si za cíl podnítit diskuzi v oblasti vizualizace agentů v ABAsim architektuře a nabídnout nové možnosti pro lepší vizualizaci.

8 Použitá literatura

[1] KAVIČKA, A., KLIMA, V., ADAMKO N.: Agentovo orientovaná simulacia dopravných uzlov. Žilina: EDIS, 2005. 206 s. ISBN: 80-8070-477-5

[2] ZBOŘIL, František. *PLÁNOVÁNÍ A KOMUNIKACE V MULTIAGENTNÍCH SYSTÉMECH*. [s.l.], 2004. 92 s. , Axiomizace CTL logiky, Komunikační akty KQML a ACL jazyků, Některé programové konstrukce v jazyce t-Sapi. VUT Brno. Fakulta Informačních techologií. Vedoucí dizertační práce doc. Dr. Ing. Petr Hanáček. Dostupný z WWW:<http://www.fit.vutbr.cz/~zborilf/PhD/thesis.pdf>

[3] RAPANT, Petr. Prostor v multiagentových systémech modelujících prostorové procesy. *Acta Montanistica Slovaca*. 2007, roč. 12, č. 2, s. 84-97. Dostupný z WWW:<http://actamont.tuke.sk/pdf/2007/n2/2rapant.pdf>

[4] GÁLET, Michal. *GRAFICKÁ NADSTAVBA PRO SYSTÉM ZÍSKÁVÁNÍ ZNALOSTÍ*. [s.l.], 2007. 51 s. FAKULTA INFORMAČNÍCH TECHNOLOGIÍ. VUT Brno. ÚSTAV INFORMAČNÍCH SYSTÉMŮ . Vedoucí diplomové práce Doc. Ing. Jaroslav Zendulka, CSc. Dostupný z WWW:<http://www.fit.vutbr.cz/study/DP/rpfile.php?id=3835>

[5] LEKÝR, Michal. *SOFTVÉROVÉ PROSTREDIE*. Žilinská univerzita v Žiline, 2006. 93 s. , obr. příl. Fakulta riadenia a informatiky. Žilinská univerzita v Žiline. Katedra dopravných sietí. Vedoucí dizertační práce Doc. Mgr. Valent Klima, Csc.

[6] KUBÍK Aleš. *Agentově-orientované inženýrství: nové paradigma pro tvorbu softwaru*. [On-line] Dostupné z WWW: <http://objekty.pef.czu.cz/2003/sbornik/Kubik2003.pdf>

[7] NETRVALOVÁ, Arnoštka. *Úvod do problematiky multiagentních systémů*. [On-line] Dostupné z WWW:<www.kiv.zcu.cz/~netrvalo/phd/MAS.pdf >

[8] BOUDREAU, Tim, TULACH, Jaroslav, WIELENGA, Geertjan. *Rich Client Programming : Plugging into the NetBeansTM Platform*. [s.l.] : Prentice Hall, 2007. 637 s. ISBN 0-13-235480-2.

[9] MIKULECKÝ, Petr, OLŠEVIČOVÁ, Kamila. *Inteligentní prostředí jako multi-agentový systém*. [On-line] Dostupné z WWW:<http://hilbert.chtf.stuba.sk/KUZVII/abstracts/Mikulecky +Olsevicova.pdf>

[10] BEČVÁŘ, Petr, KOUT, Jan, PĚCHOUČEK, Michal. *Multiagentní řízení, simulace a plánování výroby*. [On-line] Dostupné z WWW:<http://www.certicon.cz/fileadmin/Certicon/downloads/researchArticles/2_3.pdf>

[11] DĚRGEL, Pavel, FUKS, Petr. *Simulace silniční infrastruktury s využitím distribuovaných celulárních automatů v multi-agentovém systému*. [On-line] Dostupné z WWW:<http://actamont.tuke.sk/pdf/2007/n1/9dergel.pdf>

[12] ŠUŠKO, Petr. *Prehľad multiagentových systémov reaktívnych agentov*. [On-line] Dostupné z WWW:<http://neuron.tuke.sk/~susko/school/esej/esej.pdf>

[13] ŠŤASTNÝ, Pavel. *Multiagentní systémy v medicíně*. [s.l.], 2007. 66 s. , A Obsah přiloženého CD, B Poznámky k realizaci. ČVUT Praha, Fakulta elektrotechnická. Vedoucí diplomové práce Doc. Ing. Lenka Lhotská, CSc. Dostupný z WWW: <https://dip.felk.cvut.cz/browse/pdfcache/stastp2_2007dipl.pdf>

[14] HABIBALLA, Hashim. *UMĚLÁ INTELIGENCE*. [s.l.], 2004. 81 s. Přírodovědecká fakulta, Univezita Ostrava, KIP/UMINT. Distanční studijní opora. Dostupný z WWW:<http://www.volny.cz/habiballa/publ/umint.pdf>

[15] Visual Library 2.0 Documentation. Dostupný z WWW:<http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/doc-files/documentation.html>

[16] Interface List. Dostupný z WWW:<http://java.sun.com/j2se/1.4.2/docs/api/java/util/List.html>

[17] Class Vector. Dostupný z WWW:<http://java.sun.com/j2se/1.4.2/docs/api/java/util/Vector.html>

[18] Class Widget. Dostupný z WWW:<http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/Widget.html#removeChild(org.netbeans.api.visual.widget.Widget)>

[19] Class Scene. Dostupný z WWW:<http://bits.netbeans.org/dev/javadoc/org-netbeans-api-visual/org/netbeans/api/visual/widget/Scene.html>

[20] Java tutoriál - Objekty a třídy (7. díl). Dostupný z WWW:<http://programujte.com/index.php?akce=clanek&cl=2007082501-java-tutorial-objekty-a-tridy-7-dil->

[21] ŠTROBL, Roman. *Building Applications With NetBeans Visual Library*. Dostupný z WWW:<http://www.javalobby.org/eps/netbeans_visual_library/>

9 Zdroje obrázků

(1) RAPANT, Petr. Prostor v multiagentových systémech modelujících prostorové procesy. *Acta Montanistica Slovaca*. 2007, roč. 12, č. 2, s. 84-97. Dostupný z WWW: <http://actamont.tuke.sk/pdf/2007/n2/2rapant.pdf>.

(2) KAVIČKA, A., KLIMA, V., ADAMKO N.: Agentovo orientovaná simulacia dopravných uzlov. Žilina: EDIS, 2005. 206 s. ISBN80-8070-477-5

(3) LEKÝR, Michal. *SOFTVÉROVÉ PROSTREDIE*. Žilinská univerzita v Žiline, 2006. 93 s. , obr. příl. Fakulta riadenia a informatiky. Žilinská univerzita v Žiline. Katedra dopravných sietí. Vedoucí dizertační práce Doc. Mgr. Valent Klima, Csc.

(4) *NetBeans Visual Library 2.0 - Screenshots* [online]. NetBeans, 2008 [cit. 2008-06-12]. Dostupný z WWW: <http://graph.netbeans.org/screenshots.html>.