Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Výpočetní jádro pro agentově orientovanou architekturu simulačních modelů

Bc. Václav Bláhovec

Diplomová práce

2010

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2009/2010**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Václav BLÁHOVEC**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Výpočetní jádro pro agentově orientovanou architekturu simulačních modelů**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování:

1. V úvodní části práce je nutné provést popis vybrané agentově orientované architektury simulačních modelů, pro niž bude vytvářeno výpočetní jádro. 2. Primárním cílem diplomové práce je realizace efektivní implementace výpočetního jádra pro agentovou simulaci včetně animační podpory a uložení stavu příslušného simulujícího systému. 3. Výpočetní jádro bude nejdříve otestováno na vhodném jednoduchém simulačním modelu vybraného obslužného systému. 4. V další fázi testování bude simulační jádro využito pro simulační model zaměřený na zkoumání problematiky přidělování nástupištních kolejí v podmínkách zjednodušeného provozu osobní železniční stanice.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

1. KAVIČKA, Antonín, KLIMA, Valent, ADAMKO, Norbert. Agentovo orientovaná simulácia dopravných uzlov. Žilina: EDIS - vydavateľstvo ŽU, 2005. 206 s. Monografie. ISBN 80-8070-477-5. 2. ADAMKO, Norbert, KAVIČKA, Antonín, KLIMA, Valent. DAAAM International Scientific Book 2007. Branko Katalinic. Vienna : DAAAM International Publishing, 2007. ISBN 3-901509-60-7. Agent based simulation of transportation logistic systems, s. 407-422. 3. BAŽANT, Michael, KAVIČKA, Antonín. Artificial neural network as a support of platform track assignment within simulation models reflecting passenger railway stations. Journal of Rail and Rapid Transit. 2009, vol. 223, no. 5, s. 505-515.

Vedoucí diplomové práce: **doc. Ing. Antonín Kavička, Ph.D.**
Katedra softwarových technologií

Datum zadání diplomové práce: **30. října 2009**

Termín odevzdání diplomové práce: **21. května 2010**

prof. Ing. Simeon Karamazov, Dr.
děkan

L.S.

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 10. listopadu 2009

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 20. 8. 2010

Bc. Václav Bláhovec

Anotace

Diplomová práce se zabývá vytvořením efektivního výpočetního jádra pro agentově orientovanou architekturu simulačních modelů s podporou on-line animace a simulačního modelu osobní železniční stanice. V teoretické části je popsána agentově orientovaná architektura simulačních modelů. V praktické části je popsána implementace výpočetního jádra a simulačního agentově orientovaného modelu osobní železniční stanice.

Klíčová slova

simulace, agent, simulační model, MPE/ABAsim

Title

Simulation engine for agent oriented architecture of simulation models

Annotation

The thesis deals with the development of the effective simulation engine for agent oriented architecture of simulation models with the support of the on-line simulation and the development of the railway station simulation model transporting passengers only. The theoretical part is devoted to agent oriented architecture of simulation models. In the practical part there is a description of the implementation of simulation engine and agent oriented the railway station simulation model transporting passengers only.

Key words

simulation, agent, simulation model, MPE/ABAsim

Obsah

Úvod **10**
1 Cíle diplomové práce **11**
2 Architektura simulačního modelu **11**
2.1 Paradigma agentů 11
2.2 Reaktivní agenti 13
2.3 Proč používat agenty? 14
2.4 Agentově orientovaná architektura simulačního modelu 14
2.4.1 Dekompozice agenta 17
2.4.2 Vrstvový model MPE/ABAsim 19
2.4.3 Komunikační mechanizmus 20
3 Synchronizace simulačního výpočtu **23**
3.1 Synchronizace diskrétní simulace 23
3.2 Synchronizace kombinované simulace s podporou animace 24
4 Výpočetní jádro pro agentově orientovanou architekturu simulačního modelu **28**
4.1 Simulační model osobní železniční stanice 28
4.1.1 Požadavky 29
4.1.2 Zjednodušující předpoklady 29
4.2 Výběr programovacího jazyka a vývojového prostředí 30
4.3 Konceptuální model 30
4.4 Vstupní data 33
4.4.1 Infrastruktura 33
4.4.2 Vlaky 36
4.5 Simulační model 40
4.5.1 Agent okolí 41
4.5.2 Agent dispečer 42
4.5.3 Agent správa kolejiště 43
4.5.4 Agent pohyby 45
4.5.5 Agent obsluha 46
4.6 Komunikace 48
4.6.1 Zprávy 49
4.6.2 Animační zprávy 50
4.7 Životní cyklus zpracování vlaku 51
4.7.1 Fáze plánování vlaku 52
4.7.2 Fáze přidělení koleje 53
4.7.3 Fáze příjezd vlak 55
4.7.4 Fáze obsluha vlaku 58
4.7.5 Fáze odjezd vlaku 60
4.8 Vybraná implementační řešení 62
4.8.1 Infrastruktura 62

4.8.2 Algoritmus hledání nejkratší cesty v kolejišti....64
4.8.3 Řídící algoritmus diskrétní simulace s on-line animací....68
4.9 Možnosti rozšíření....70
Závěr....71
Obsah přiloženého datového média....72
Seznam použité literatury....73

Seznam obrázků

Obr. 2.1: Klasifikace agentů [4].....13
Obr. 2.2: Funkce agenta [4].....15
Obr. 2.3: Schopnosti agenta [4].....16
Obr. 2.4: Dekompozice agenta [4].....19
Obr. 2.5: Vrstvy simulačního modelu [4].....20
Obr. 3.1: Struktura kombinovaného simulačního modelu s animátorem [4].....25
Obr. 3.2: Přidělování časových kvant [4].....27
Obr. 4.1: První konceptuální model (Zdroj: vlastní).....31
Obr. 4.2: Druhý konceptuální model (Zdroj: vlastní).....32
Obr. 4.3: Simulační model (Zdroj: vlastní).....40
Obr. 4.4: Vnitřní struktura agenta okolí (Zdroj: vlastní).....41
Obr. 4.5: Vnitřní struktura agenta dispečera (Zdroj: vlastní).....42
Obr. 4.6: Vnitřní struktura agenta správa kolejiště (Zdroj: vlastní).....43
Obr. 4.7: Vnitřní struktura agenta pohyby (Zdroj: vlastní).....45
Obr. 4.8: Vnitřní struktura agenta obsluha (Zdroj: vlastní).....46
Obr. 4.9: Zjednodušený vrstvový MPE/ABAsim model osobní železniční stanice (Zdroj: vlastní).....48
Obr. 4.10: Komunikační diagram plánování vlaku (Zdroj: vlastní).....52
Obr. 4.11: Komunikační diagram přidělení koleje (Zdroj: vlastní).....54
Obr. 4.12: Komunikační diagram příjezdu vlaku (Zdroj: vlastní).....57
Obr. 4.13: Komunikační diagram obsluhy vlaku (Zdroj: Vlastní).....59
Obr. 4.14: Komunikační diagram odjezdu vlaku (Zdroj: vlastní) 61
Obr. 4.15: Znázornění vnitřní struktury koleje (Zdroj: vlastní).....62
Obr. 4.16: Znázornění vnitřní struktury a) křížení b) jednoduché výhybky c) poloviční anglické výhybky d) anglické výhybky (Zdroj: vlastní).....63
Obr. 4.17: Ukázka části zjednodušeného grafu s vyznačením vnitřní struktury prvků (Zdroj: vlastní).....63
Obr. 4.18: Ukázka špatně nalezených cest (Zdroj: vlastní).....64
Obr. 4.19: Zpracování zakázaného odbočení na výhybce (Zdroj: vlastní).....66
Obr. 4.20: Přidělování časových kvant mezi modulem diskrétní simulace a animace (Zdroj: vlastní).....68

Seznam tabulek

Tab. 3.1: Synchronizační algoritmus využívající přímé i centrální doručování [4] 24
Tab. 3.2: Řídící algoritmus kombinované simulace s on-line animací [4].....26
Tab. 4.1: Základní porovnání programovacích jazyků (Zdroj: vlastní).....30
Tab. 4.2: Kódy zpráv, jejich význam a použití (Zdroj: vlastní).....50
Tab. 4.3: Kódy animačních zpráv, jejich význam a použití (Zdroj: vlastní).....51
Tab. 4.4: Řídící algoritmus diskrétní simulace s on-line animací (Zdroj: vlastní) 69

Seznam použitých zkratek

Zkratka	**Význam v anglickém jazyce**	**Význam v českém jazyce**
ABAsim	**A**gent-**B**ased **A**rchitecture of **sim**ulation model	Agentově orientovaná architektura simulačního modelu.
MPE	**M**anagement-**P**rocessing-**E**ntities	Management, procesy, entity.
XML	**E**xtensible **M**arkup **L**anguage	rozšiřitelný značkovací jazyk

Úvod

V dnešní době nelze cokoliv vyvíjet, postavit či implementovat bez předchozího odzkoušení, protože pozdější úpravy by znamenaly další náklady. Proto je dnes téměř ve všech odvětvích využívána simulace jako prostředek pro ověření návrhu bez nutnosti investice nákladů do později nefunkční realizace návrhu.

Navrhnout simulační model libovolného zaměření není v dnešní době problém a k tomuto účelu existuje řada podpůrných simulačních nástrojů. Pokud ale pro návrh a vybudování simulačního modelu daného zadání neexistuje vhodný simulační nástroj, je nutné ho navrhnout a implementovat nebo upravit již existující simulační nástroj.

Při návrhu simulačního nástroje lze použít doporučené architektury, které již byly ověřeny, nebo vyvinout vlastní. Rozhodnutí o použití správné architektury simulačního modelu je velmi důležité, protože se od ní odvíjí použité technologie, návrh simulačního modelu apod. Jednou z těchto architektur je např. ABAsim architektura.

Tato diplomová práce se zabývá návrhem a implementací efektivního výpočetního jádra pro agentově orientovanou architekturu simulačních modelů s podporou on-line animace a simulačního modelu osobní železniční stanice. V teoretické části je popsána agentově orientovaná architektura simulačních modelů. V praktické části je popsána implementace výpočetního jádra a simulačního agentově orientovaného modelu osobní železniční stanice.

1 Cíle diplomové práce

Cílem této diplomové práce je efektivní implementace výpočetního jádra pro agentově orientovanou architekturu simulačních modelů s podporou on-line animace. Výpočetní jádro bude implementované jako knihovna, která bude použita pro implementaci simulačního nástroje [5] podporujícího on-line animaci.

Dalším cílem této diplomové práce je implementace simulačního agentově orientovaného modelu osobní železniční stanice a otestování na datech osobní železniční stanice Praha hlavní nádraží.

2 Architektura simulačního modelu

Simulační model rozsáhlého systému, jako je železniční stanice, je velmi složitý a jeho implementace značně náročná. Proto je zapotřebí celý systém dobře strukturovat, aby se nechal jednoduše spravovat, ladit a dále rozšiřovat bez nutnosti zásahu do základní koncepce simulačního modelu.

2.1 Paradigma agentů

Koncepce agentů má základy v umělé inteligenci a v současné době je využívána v řadě oborů informačních technologií (např. web-technologie, robotika, simulace).

Agent je zapouzdřený počítačový systém zasazený do nějakého prostředí, kde pružně a autonomně působí se záměrem plnění daného cíle, přičemž za jeho klíčové vlastnosti považujeme:

- **autonomnost**, tj. agent je schopný pracovat samostatně bez vnějších zásahů a úplně řídit svoje výkony a kontrolovat svůj vnitřní stav,
- **společenské chování**, které se projevuje jako interakce s jinými agenty (resp. s člověkem) prostřednictvím jistého komunikačního mechanizmu (jazyka),

- **reaktivitu** anebo reagování na podměty z okolního prostředí a
- **iniciativnost**, tzn. že agent nereaguje jen na podměty z okolního prostředí, ale je schopný se chovat cíleně vyvíjením vlastní iniciativy (podporované schopností učit se) [2].

Pojem „zapouzdřený počítačový systém“ lze chápat jako realizaci agenta čistě softwarově nebo pomocí hardwarových komponentů.

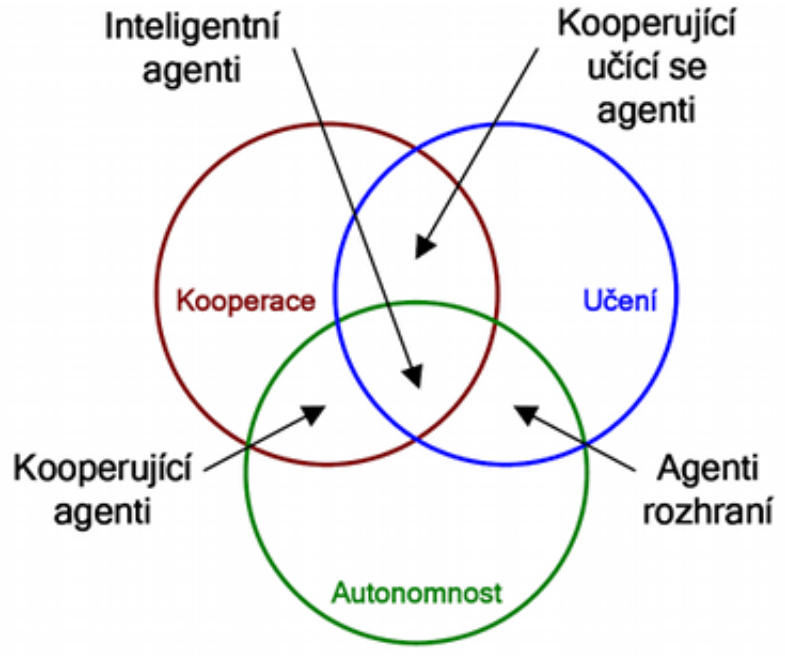
Při implementaci (technické realizaci) agenta, resp. celého společenství agentů, se používají různé koncepce, které kladou různý důraz na výše uvedené vlastnosti, vhodné pro implementaci v různých prostředích. Každá z koncepcí vychází z aktuálních potřeb implementace agentů v aplikacích nebo ze vzoru, definujícího řešený problém.

Agenty lze klasifikovat:

- Agenty lze klasifikovat podle jejich *mobility*, tj. schopnosti jejich pohybu v rámci počítačových sítí, a tedy rozdělovat je na *statické* („žijících jen na jednom počítači“) a *mobilní agenty*, kteří se můžou v síti přemisťovat.
- Dalším klasifikačním kritériem je *míra iniciativnosti agenta*, přičemž z tohoto pohledu rozlišujeme *uvažující* a *reaktivní agenty*. Rozhodnutí uvažujících agentů (např. o tom, jakou činnost je potřeba v daném okamžiku vykonat) jsou založená na logickém uvažování a existenci explicitně definovaného interního modelu okolního prostředí agenta. Uvažující agenti tedy můžou sami kdykoliv rozhodnout o potřebě vykonání jisté činnosti. Naproti tomu pro reaktivní agenty je typické, že pracují na principu podnět→odezva, tedy reagují na aktuální stav prostředí, v kterém jsou začleněni (nedisponují však žádným interním modelem svého prostředí). Všechny činnosti reaktivních agentů jsou tedy jen reakcí na podnět z okolí.

- Jiné možní členění je založené na různých kombinacích důrazu (uplatněného při konkrétní realizaci agenta) na hlavní vlastnosti agenta, tedy na *autonomnost*, *kooperativnost* a *schopnost učit se*. Při tomto typu členění tedy rozlišujeme *kooperativní agenty*, *agenty rozhraní*, *kooperativní učící se agenty* a *inteligentní agenty* [3].

Na obr. 2.1 jsou symbolicky graficky znázorněné vzájemné průniky uvedených vlastností, které reprezentují různé kategorie agentů.



Obr. 2.1: Klasifikace agentů [4]

2.2 Reaktivní agenti

Reaktivní agent vykonává svoji činnost pouze na základě podnětů ze svého okolí (nedisponuje žádným popisem svého okolí). Při konstrukci reaktivního agenta se nevytváří žádný plán jeho budoucího chování, to vyplývá až z aktuálního stavu jeho okolí. Někdy se reaktivní agenti označují jako situační agenti právě z důvodu reagování na danou situaci.

Agent je sestaven z kolekce modulů, které vykonávají svoji činnost samostatně a jsou určeny pro řešení konkrétní úlohy (např. výpočty, zjišťování stavu systému

apod.). Tyto moduly mezi sebou komunikují jen minimálně a to na velmi nízké úrovni. Žádný agent nedisponuje modelem celého systému, tudíž jeho chování vyplyne až po začlenění do většího celku.

Svou činností reaktivní agenti připomínají více senzorické systémy než činnost systému s prvky umělé inteligence. Inteligence společenství reaktivních agentů se objeví až sledováním celého společenství jako celku a ne jednotlivých agentů.

2.3 Proč používat agenty?

Systém složený ze společenství agentů má několik předností oproti systému vytvořenému jako jeden celek. Těmito výhodami jsou vysoký stupeň decentralizace, systém nevyžaduje návrh centrálního řízení, a poměrně vysoká míra životaschopnosti jednotlivých členů společenství plynoucích z jejich autonomní podstaty. Díky těmto vlastnostem lze postavit systém složený z mnoha navzájem spolupracujících jedinců bez nutnosti definování přesného chování celého společenství (např. jednotky řízení ve složitých obslužných systémech nebo jednotlivého jedince biologických společenstev).

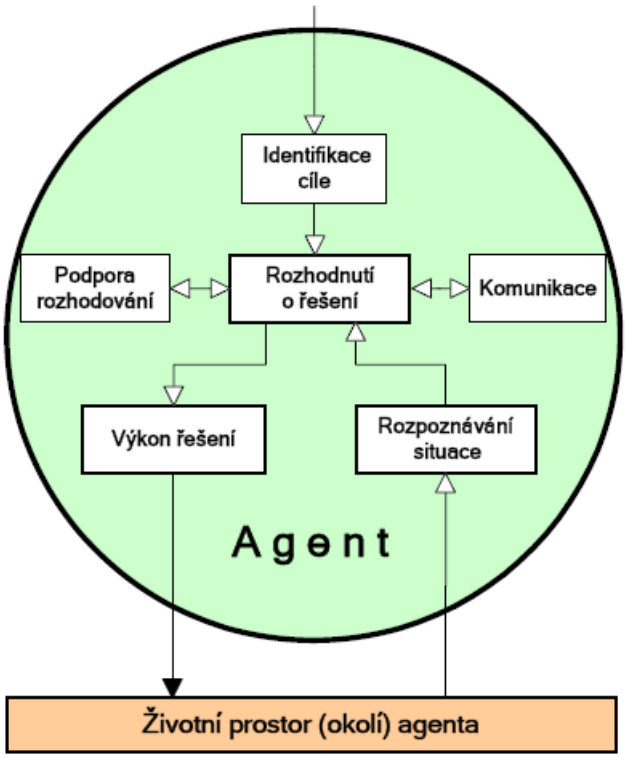
2.4 Agentově orientovaná architektura simulačního modelu

Definovat pojem architektura simulačního modelu není jednoduché a jednoznačné, lze však uvést základní faktory, které architekturu simulačního modelu charakterizují:

- základní stavební kameny, tj. základní strukturální nebo funkčně-akční jednotky, ze kterých je simulační model postavený (např. událost, aktivita, proces, agent apod.),
- rozdělení (alokace) simulačního výpočtu na výpočetní jednotky (procesory, resp. počítače). V této souvislosti hovoříme o *monolitické architektuře* (realizující simulační model na jednoprocesorovém počítači)

nebo o *paralelní*, resp. *distribuované architektuře*, která je realizovaná na víceprocesorovém počítači, resp. na více počítačích v počítačové síti [1].

Na obr. 2.2 jsou zachyceny základní funkce agenta. Úkolem každého agenta je splnit zadaný nebo identifikovaný cíl. Zadaný cíl je agentovi určen již při jeho realizaci a je jasné, čeho tento agent musí dosáhnout. Naproti tomu identifikování cíle je doménou inteligentních agentů, kdy agent sám musí tento cíl identifikovat na základě svých schopností.

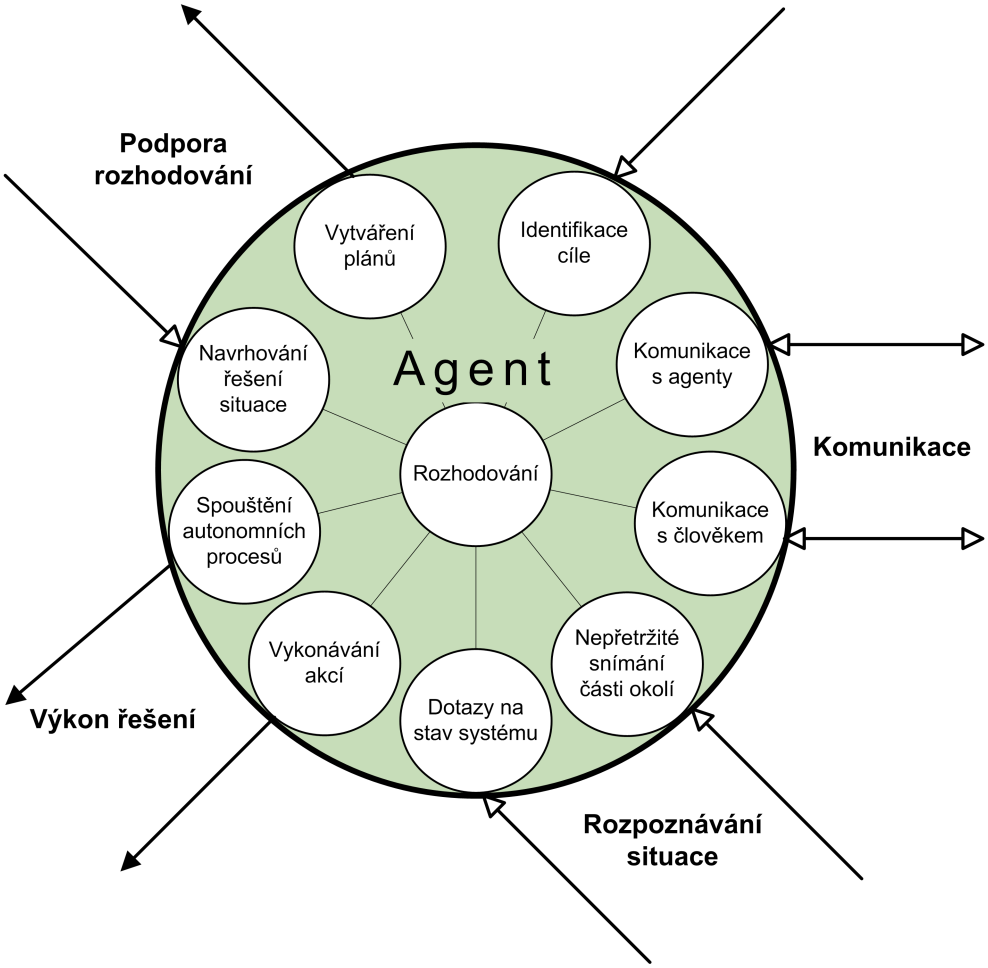


Obr. 2.2: Funkce agenta [4]

Každý agent realizuje svůj životní cyklus: *rozpoznání situace* – *rozhodování o řešení* – *výkon řešení* v daném prostředí, kde agent působí. Prostředí, do něhož je agent zasazen, zahrnuje všechny ostatní agenty, se kterými daný agent komunikuje, a příslušnou část stavového prostoru, ve které je agent oprávněn vykonávat změny. Dále může být toto prostředí jednoznačně vymezeno

(v případě statických agentů) nebo se může měnit (to platí především pro mobilní agenty).

Jako podpora rozhodování agentovi slouží funkce *návrh řešení problému* a *komunikace* s ostatními agenty (případně s člověkem). Je-li identifikována situace, kterou agent nedokáže vyřešit (není v jeho kompetenci), informuje o tom ostatní agenty a tím přispěje k nalezení řešení.



Obr. 2.3: Schopnosti agenta [4]

Aby agent mohl vykonávat výše uvedené funkce, musí mít určité vlastnosti (obr. 2.3). Pro komunikace s jinými agenty, resp. s člověkem, má k dispozici jistý komunikační mechanizmus. Pro podporu rozhodování má agent schopnost navrhnout jednorázové řešení problémů (typicky jsou to volání

optimalizovaných algoritmů), jako i schopnost vytvářet dlouhodobější plány vykonávání určitých činností (technologické postupy, rozvrhování zdrojů apod.). Schopnost jednorázově se zeptat na aktuální stav systému nebo snímat stav systému v určitých intervalech představuje senzorickou funkci agenta. Funkci změny stavu systému představují schopnosti okamžité změny stavu systému (aktivace akce) nebo provádění změn bez vnějšího zásahu v určitých časových intervalech (spuštění samostatného procesu).

Výhodné je rozdělení schopností agenta do skupin a každou schopnost implementovat jako nezávislý modul. Tato dekompozice agenta přinese jisté výhody:

- každý takovýto modul (komponent) lze použít ve více agentech najednou,
- možnost implementace více alternativ jediného komponentu, ze kterých si poté uživatel může vybrat takový komponent, který je nejlepší pro daný simulační experiment.

2.4.1 Dekompozice agenta

Agenta je vhodné rozdělit například na čtyři skupiny interních komponentů (obr. 2.4).

I. První skupina *řídících a rozhodovacích komponentů* obsahuje komponent nazvaný *manažer*, který přebírá schopnosti agenta pro identifikaci cíle, komunikace a rozhodování. Manažer má speciální funkci jako ústřední komponent. Dokáže komunikovat s ostatními komponenty a některé i řídit. Ostatní interní komponenty mezi sebou nekomunikují.

II. Skupina *senzorů* zpřístupňuje informace o stavu systému a obsahuje komponenty dvou druhů: *dotaz* a *monitor*. Komponent *dotaz* zprostředkovává informace manažerovi okamžitě na jeho pokyn.

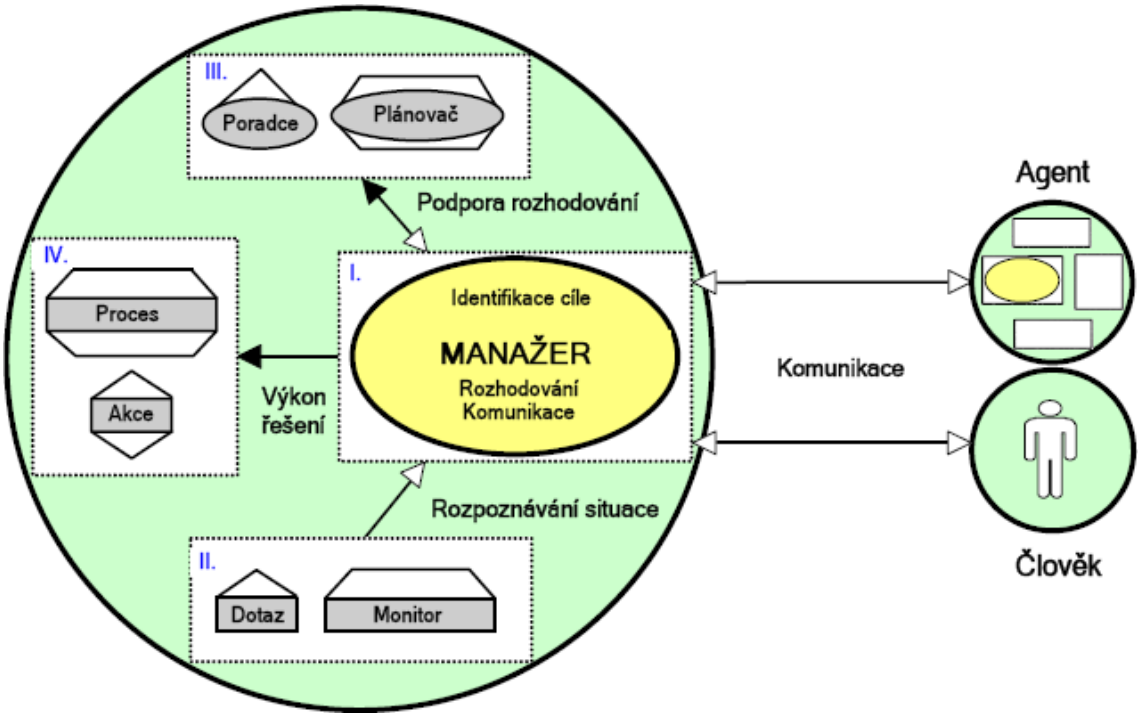
Naproti tomu komponent *monitor* snímá informace ze stavového prostoru v časových intervalech.

III. Další skupinou je skupina *řešitelů*. Do této skupiny lze zařadit komponenty *poradce* a *plánovač*. Poradce je pasivní komponent zprostředkovávající okamžitý návrh (návrhy) řešení na pokyn manažera. Poradcem může být optimalizační algoritmus nebo člověk. Plánovač předpovídá možné problémy, resp. konflikty a snaží se pro určité časové intervaly vytvořit časový plán pro jejich řešení. Plánovač tyto časové plány aktualizuje bez zásahu manažera a pokud identifikuje závažný problém, informuje o něm manažera.

IV. Poslední skupinou jsou *efektory*, které jako jediné komponenty můžou měnit stav systému. Do této skupiny se řadí komponent *akce* a *proces*. Komponent *akce* vykoná změnu v tom jistém okamžiku, kdy byla iniciovány manažerem. Proces zajišťuje změnu stavu systému v jistých časových intervalech (např. realizace pohybů, změna signalizace apod.).

Komponenty efektory, senzory a řešitele lze označit společným názvem asistenti (pomocníci manažera) a je možné je dělit na:

- *kontinuální asistenty*, jejichž činnost trvá nenulový simulační čas (procesy, monitory a plánovače) a
- *promptní asistenty*, jejichž činnost se vyková v jednom okamžiku simulačního času (akce, dotazy a poradci).



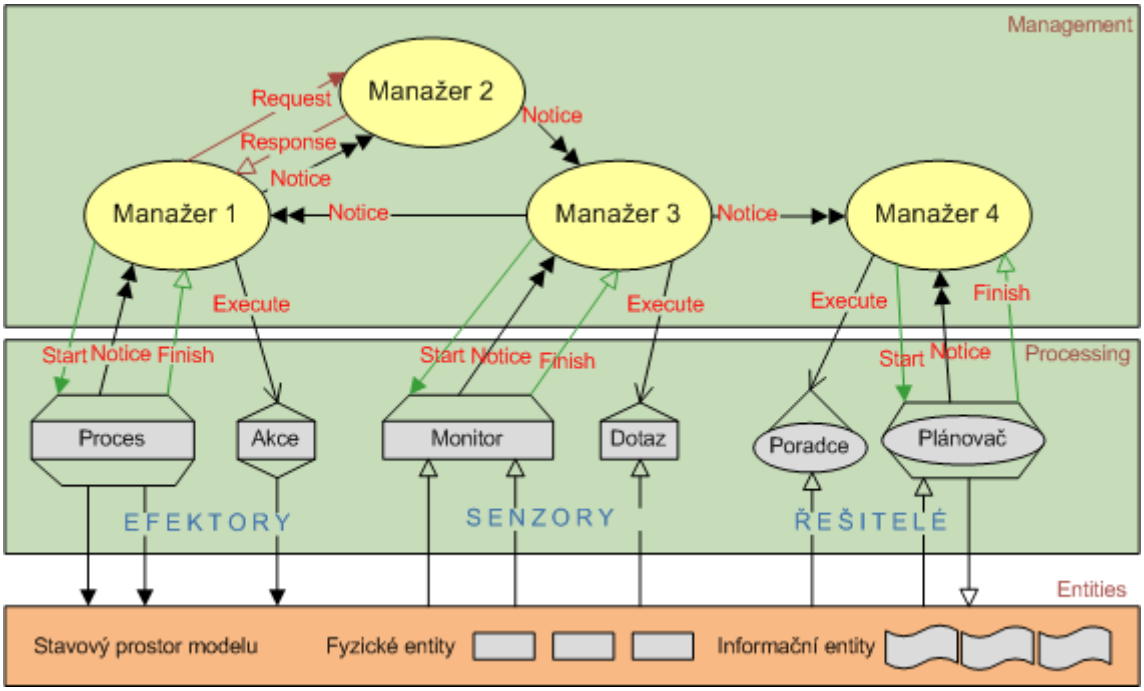
Obr. 2.4: Dekompozice agenta [4]

2.4.2 Vrstvový model MPE/ABAsim

Po dekompozici agentů na dvě základní skupiny interních komponentů (manažeři a asistenti) se lze na simulační model podívat jako na systém složený z několika vrstev (obr. 2.5):

- *vrstvy managementu*, v které se přijímají rozhodnutí a vydávají pokyny k jejich realizaci,
- *vrstvy zpracování* (processing), kde komponenty asistují managementu a realizují změny ve stavovém prostoru simulačního modelu,
- *vrstvy entit*, která vytváří stavový prostor simulačního modelu. V této vrstvě se nacházejí *fyzické* a *informativní entity*. Fyzické entity jsou zobrazením všech prvků simulovaného modelu a jejich souhrn odráží aktuální stav simulačního modelu. Informační entity neobsahují atributy fyzických entit, ale zprostředkovávají informace pro řízení systému nebo informace popisující průběh simulace (např. statistické údaje).

Tento vrstvový model lze zkráceně označit jako MPE/ABAsim (**M**anagement-**P**rocessing-**E**ntities/**A**gent-**B**ased **A**rchitecture of **sim**ulation model) a vyplývá z něj, že úkolem manažerů je vzájemná spolupráce a za pomoci senzorů a řešitelů startovat činnost efektorů ve správném čase a při splnění potřebných podmínek.



Obr. 2.5: Vrstvy simulačního modelu [4]

2.4.3 Komunikační mechanizmus

V ABAsim architektuře je komunikace realizovaná pouze pomocí zasílání zpráv, proto je označována jako *zprávově orientovaná*. Komunikace probíhá na dvou úrovních a to mezi agenty (komunikuji spolu jednotliví manažeři) a uvnitř agenta (manažer komunikuje se svými asistenty). V následujícím přehledu jsou uvedeny typy zpráv, které lze použít pro komunikaci.

I. Pro komunikaci mezi agenty (mezi jejich manažery) se používají tři 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 na ni očekává od adresáta odpověď.
   - **Response** (reakce/odpověď), která reprezentuje povinnou odpověď na zprávu typu Request, přičemž může být doručena pouze odesílateli.

II. Pokyny, které může manažer dávat svým asistentům, jsou realizované pomocí následujících typů zpráv:
   - **Start**, po přijetí zprávy tohoto druhu začne adresát (musí to být kontinuální asistent) vykonávat svoji autonomní činnost.
   - **Break** představuje jediný způsob, jak může manažer ovlivňovat autonomní běh kontinuálního asistenta, který je po přijetí této zprávy nenávratně ukončen (tento typ zprávy se používá jen ve výjimečných případech, kdy nastaly v simulačním modelu takové podmínky, při kterých už nemá smysl, aby kontinuální asistent dokončil svoji činnost).
   - **Execute** je specifickým typem zprávy určeným pro promptního asistenta, který okamžitě vykoná odpovídající činnost a obratem zprávu vrátí svému manažerovi (odesílateli) s vyznačeným výsledkem své činnosti.

III. Posledním druhem zpráv, které se v ABAsim architektuře používají jsou zprávy, které odesílají kontinuální asistenti během své autonomní činnosti:

   - **Finish** je zpráva, kterou každý z kontinuálních asistentů pošle svému manažerovi v okamžiku ukončení své činnosti.
   - **Notice** jsou zprávy, které můžou být poslané v zásadě kdykoliv v průběhu činnosti kontinuálního asistenta, když kontinuální asistent potřebuje oznámit svému manažerovi výskyt nějaké pro něj důležité situace.
   - **Hold** je jedinou zprávou, která obsahuje tzv. *časové razítko* určující požadovaný čas doručení, přičemž tuto zprávu si kontinuální asistent posílá zásadně sám sobě (kontinuální asistent může po odeslání této zprávy opět pokračovat ve své činnosti až do toho okamžiku simulačního času, kdy je mu tato zpráva doručena – do té doby je neaktivní) [4].

Z předešlého výčtu typu zpráv je patrně, že plynutí simulačního času zajišťují výhradně zprávy typu *Hold* a tedy časová synchronizace modelu vlastně představuje synchronizaci činností kontinuálních asistentů. Ve vrstvě managementu se zprávy s časovým razítkem nepoužívají a tudíž zde „neplyne“ simulační čas. To platí pouze pro monolitickou realizaci (na jednom počítači) ABAsim architektury. Pro distribuovanou simulaci by museli mít i zprávy zasílané ve vrstvě managementu časové razítko.

3 Synchronizace simulačního výpočtu

3.1 Synchronizace diskrétní simulace

Synchronizace diskrétní simulace v ABAsim architektuře tedy probíhá pomocí zasílání zpráv určitého typu (Hold). Zprávu lze chápat jako datovou strukturu – formulář, do jehož políček se zapisují údaje důležité pro komunikaci mezi agenty, resp. komponentami. Takže zpráva musí uchovávat informace o *typu zprávy* (např. Notice, Start, Hold), o *odesílateli* a *adresátovi*, dále o *času doručení* (pokud je to zpráva označená časovým razítkem) a nakonec o vlastním obsahu zprávy, který se může skládat z *kódu zprávy* (např. „Zdroj obsluhy přidělen“) a z případného *výsledku* výpočtu, který zpráva nese (např. referenci na přidělený zdroj obsluhy apod.).

V architektuře MPE/ABAsim je velké množství zpráv, které nemají časové razítko a tudíž jsou zpracovávané tom samém okamžiku simulačního času, v kterém byli odeslané. Proto byla navržena a implementována synchronizační metoda, která umožňuje dva různé způsoby doručování zpráv a to *centrální doručování* (zprávy označené časovým razítkem prochází přes centrální poštu) a *přímé doručování* (každá komponenta má svou vlastní poštovní schránku a zprávy bez časového razítka je tedy možno doručit přímo adresátovi bez nutnosti průchodu přes centrální poštu).

Princip algoritmu synchronizační metody s centrální i přímým doručováním je založen na tom, že simulační jádro pracuje ve dvou základních fázích. V první fázi je postupně vydán pokyn všem komponentům, aby zpracovaly všechny zprávy, které mají ve svých poštovních schránkách. Tím ale mohou být vygenerovány další zprávy a proto tato fáze končí v okamžiku, kdy už všechny komponenty mají prázdné schránky.

V druhé fázi se z centrální pošty odeberou všechny zprávy s nejmenším (stejným) časovým razítkem a jsou doručeny do poštovních schránek příslušných adresátů. Poté se algoritmus vrací opět do své první fáze.

Hlavní principy tohoto synchronizačního algoritmu jsou popsány v tab. 3.1.

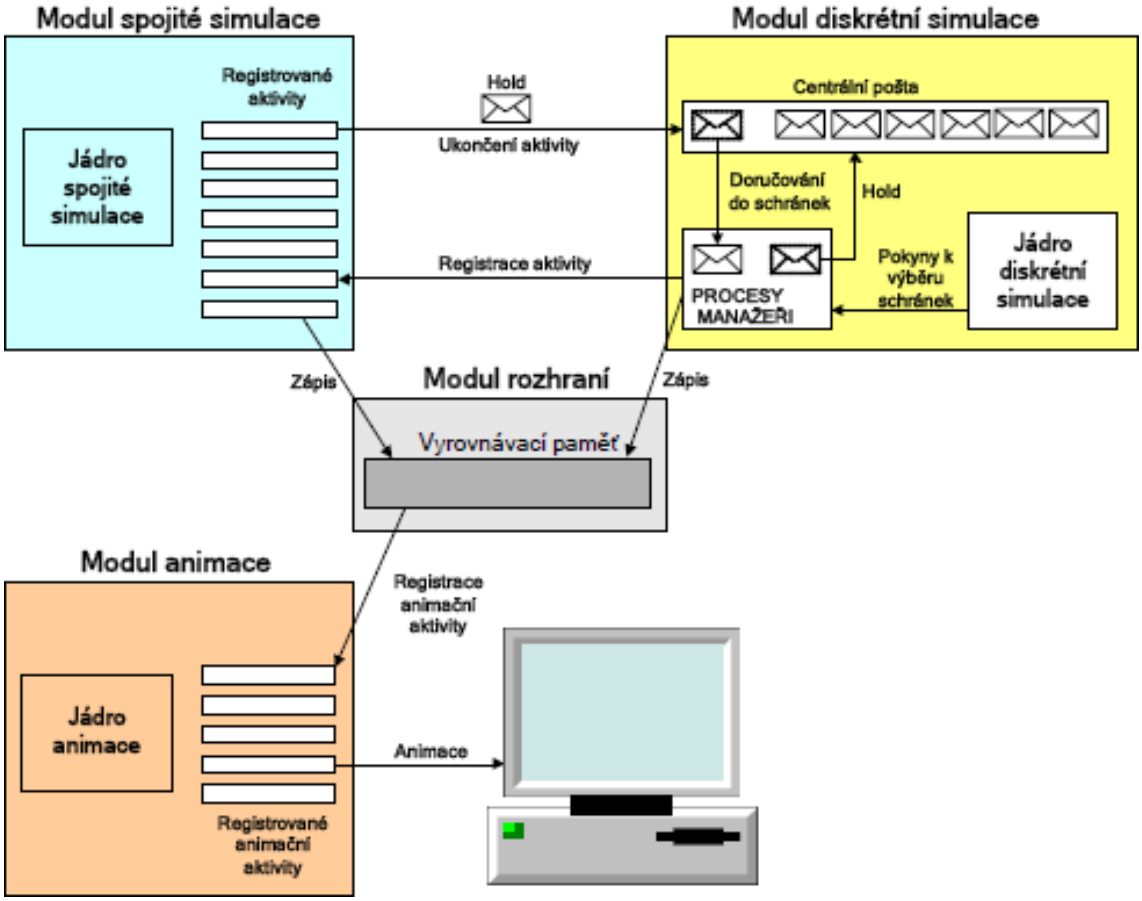
Krok	Činnost	Vykonána za podmínek
0	Inicializace simulačního času $\mathbf{t_S}$ ($\mathbf{t_S} = 0$)	
1	Zjištění celkového počtu zpráv $\mathbf{n_P}$ v poštovních schránkách všech komponentů a zjištění naplněnosti kalendáře (centrální pošty). $\mathbf{n_K} = 0$ … prázdný kalendář $\mathbf{n_K} > 0$ … neprázdný kalendář	
2	Ukončení simulace.	Platí ($\mathbf{n_P} = 0 \wedge \mathbf{n_K} = 0$) nebo je vyčerpán určený čas pro běh simulačního programu.
3	**1. fáze** Postupné vydávání pokynů všem komponentům k výběru zpráv ze svých poštovních schránek a k jejich okamžitému zpracování. Pozn.: vydávání zmíněných pokynů je ukončeno jestliže $\mathbf{n_P} = 0$.	$\mathbf{n_P} > 0$
4	**2. fáze** Aktualizuje simulačního času $\mathbf{t_S} = \mathbf{t_V}$, kde $\mathbf{t_V}$ představuje hodnotu nejmenšího časového razítka ze všech zpráv aktuálně obsažených v kalendáři (centrální poště).	$\mathbf{n_K} > 0$
5	Výběr všech zpráv z kalendáře /pošty/ s časovým razítkem $\mathbf{t_V}$ a jejich doručení do poštovních schránek příslušných adresátů.	$\mathbf{n_K} > 0$
6	Návrat na **KROK 1**.	

Tab. 3.1: Synchronizační algoritmus využívající přímé i centrální doručování [4]

3.2 Synchronizace kombinované simulace s podporou animace

Pokud nelze nějakou z aktivit vyjádřit diskrétním průběhem, je zapotřebí vytvoření modulu, který se postará a o tuto aktivitu jako aktivitu se spojitým průběhem. Proto je nutné rozšířit řízení jádra diskrétní simulace o možnost řídit i jádro spojité simulace. Navíc je vhodné zobrazovat i průběžné výsledky v průběhu simulace. K tomu se může využít on-line animace. Takže vzniká

požadavek na řízení třech různých modulů a to *modul diskrétní simulace* (stará se o aktivity s diskrétním průběhem), *modul spojité simulace* (stará se o aktivity se spojitým průběhem) a *modul animace* (stará se on-line animační výstup průběžných výsledků simulace). Dále se dá uvažovat o tzv. *modulu rozhraní*, který slouží jako vyrovnávací paměť pro animační data mezi moduly diskrétní a spojité simulace a modulem animace. Struktura kombinovaného simulačního modelu s animátorem je znázorněna na obr. 3.1.

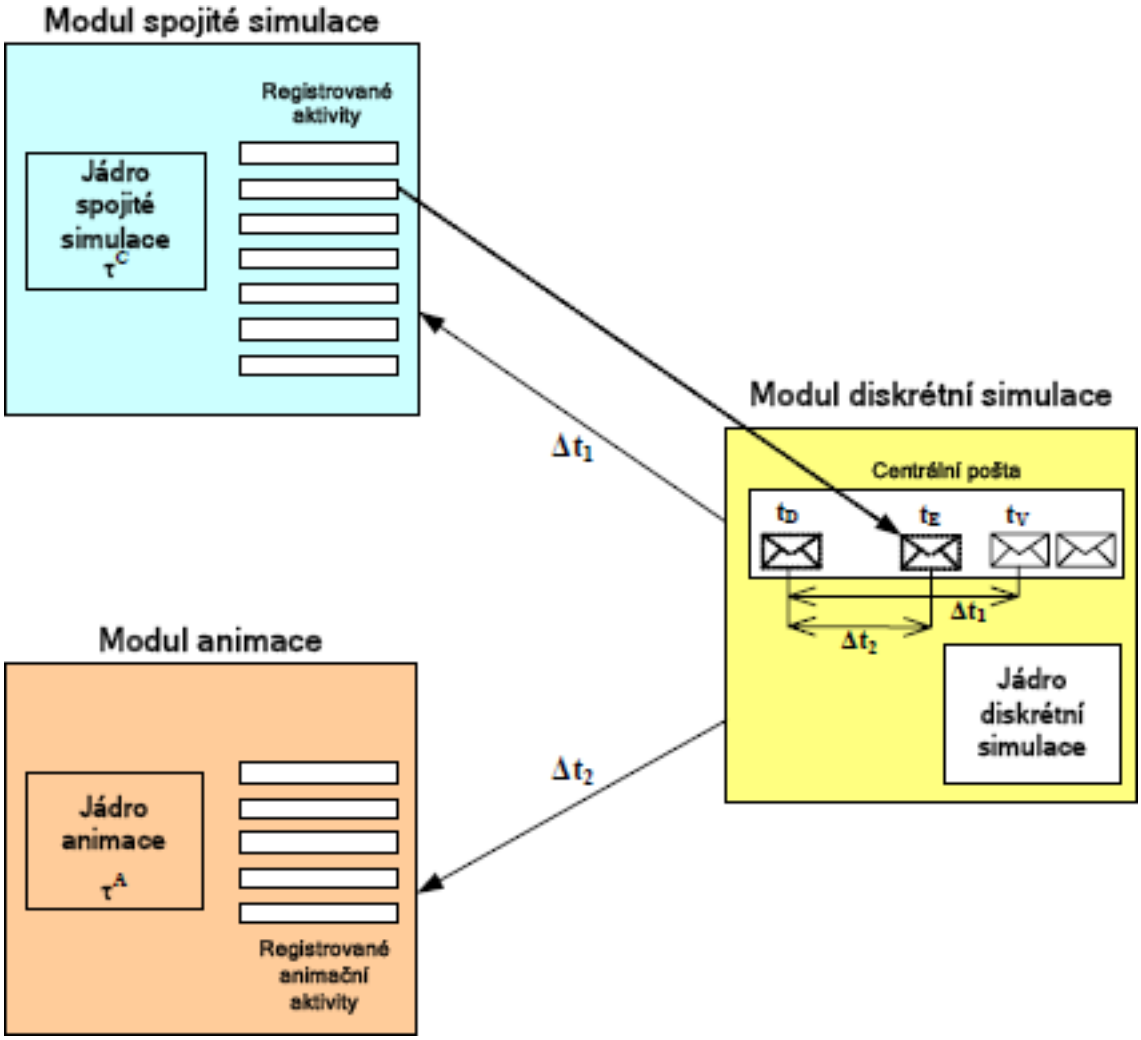


Obr. 3.1: Struktura kombinovaného simulačního modelu s animátorem [4]

Z této struktury plyne, že celé řízení probíhá pomocí modulu diskrétní simulace, kdy ve vhodných okamžicích předává řízení ostatním modulům na přesně definovaný časový interval.

Krok	Vykonává	Činnost	Vykonaná za podmínky
0	**Jádro diskrétní simulace**	Inicializace simulačního času diskrétní simulace $\mathbf{t_D}$ ($\mathbf{t_D} = 0$).	
1		Zjištění celkového počtu zpráv $\mathbf{n_P}$ v poštovních schránkách všech komponentů a zjištění naplněnosti kalendáře (centrální pošty). $\mathbf{n_K} = 0$ … prázdný kalendář $\mathbf{n_K} > 0$ … neprázdný kalendář	
2		Ukončení simulace.	Platí ($\mathbf{n_P} = 0 \wedge \mathbf{n_K} = 0$) nebo je vyčerpán určený čas pro běh simulačního programu.
3		Postupné vydávání pokynů všem komponentům k výběru zpráv ze svých poštovních schránek a k jejich okamžitému zpracování. Pozn.: vydávání zmíněných pokynů je ukončeno jestliže $\mathbf{n_P} = 0$.	$\mathbf{n_P} > 0$
4		Zjištění časového kvanta $\mathbf{\Delta t_1}$.	
5	**Jádro spojité simulace**	Aplikace metody snímání aktivit (spojitého charakteru) se snímací periodou $\boldsymbol{\tau^C}$ na všechny zaregistrované aktivity až do výskytu přerušení.	$\mathbf{\Delta t_1} \neq 0$ a počet zaregistrovaných „spojitých“ aktivit $\mathbf{n_C} \neq 0$
6	**Jádro diskrétní simulace**	Zjištění časového kvanta $\mathbf{\Delta t_2}$.	
7	**Jádro animace**	Provedení registrace animačních aktivit na základě údajů z vyrovnávací paměti.	$\mathbf{\Delta t_2} \neq 0$
8		Aplikace metody snímání animačních aktivit se snímací periodou $\boldsymbol{\tau^A}$ až do vyčerpání svého časového kvanta $\mathbf{\Delta t_2}$.	$\mathbf{\Delta t_2} \neq 0$ a počet zaregistrovaných „animačních“ aktivit $\mathbf{n_A} \neq 0$.
9	**Jádro diskrétní simulace**	Vydání pokynu k vyprázdnění vyrovnávací paměti.	Vyrovnávací paměť je neprázdná.
10		Aktualizuje simulačního času diskrétní simulace $\mathbf{t_D} = \mathbf{t_V}$, kde $\mathbf{t_V}$ představuje hodnotu nejmenšího časového razítka ze všech zpráv aktuálně obsažených v kalendáři (centrální poště).	$\mathbf{n_K} > 0$
11		Výběr všech zpráv z kalendáře (centrální pošty) s časovým razítkem $\mathbf{t_V}$ a jejich doručení do poštovních schránek příslušných adresátů.	$\mathbf{n_K} > 0$
12		Návrat na **KROK 1**.	

Tab. 3.2: Řídící algoritmus kombinované simulace s on-line animací [4]



Obr. 3.2: Přidělování časových kvant [4]

Modul diskrétní simulace provádí svou činnost až do okamžiku, kdy už nejsou ke zpracování žádné zprávy pro aktuální simulační čas t_D (zprávy se stejným časovým razítkem. Pokud jsou zpracovány všechny zprávy, je vypočítáno časové kvantum $\Delta t_1 = t_D - t_V$, kde t_V je nejmenší časové razítko ze všech zpráv v centrální poště. Je to časové kvantum, po které modul diskrétní simulace neprovádí žádnou činnost, a je možné předat řízení modulu spojité simulace. V tomto časovém kvantu provádí modul spojité simulace svou činnost za pomocí metody *snímání aktivit* s periodou τ^C a po vyčerpání tohoto kvanta předá řízení zpět modulu diskrétní simulace. Během činnosti modulu spojité simulace mohlo dojít k naplánování ukončení nějaké aktivity pomocí doručení

zprávy typu *Hold* s časovým razítkem t_E do centrální pošty a tím nevyčerpání veškerého přiděleného časového kvanta Δt_1. Dalším krokem je zjištění časového kvanta $\Delta t_2 = t_E - t_D$, které určuje dobu, po kterou lze předat řízení modulu animace. Podobně jako modul spojité simulace provádí i modul animace svou činnost pomocí metody *snímání animačních aktivit* s periodou τ^A až do doby vyčerpání přiděleného časového kvanta. Poté opět předá řízení modulu diskrétní simulace a celý cyklus se opakuje. Na obr. 3.2 je znázorněno přidělování časových kvant a v tab. 3.2 je uveden řídící algoritmus všech tří modulů.

4 Výpočetní jádro pro agentově orientovanou architekturu simulačního modelu

Hlavní náplní této diplomové práce je realizace efektivní implementace výpočetního jádra pro agentově orientovanou architekturu simulačního modelu a implementace simulačního agentově orientovaného modelu osobní železniční stanice. Toto výpočetní jádro je použito pro implementaci simulačního nástroje [5], který podporuje animaci simulovaných aktivit.

Výpočetní jádro je využitelné i pro jiné simulační modely, protože řídící algoritmus byl implementován nezávisle na simulačním modelu.

4.1 Simulační model osobní železniční stanice

Jednou z částí implementace bylo vytvoření simulačního modelu železniční stanice založeného na agentově orientované architektuře. Při návrhu a implementaci byla využita architektura vrstvového modelu MPE/ABAsim. Protože není potřeba simulovat spojité procesy, byly implementována podpora diskrétní simulace a veškeré popisované procesy mají diskrétní charakter.

Jako vstupní data simulačního modelu posloužila infrastruktura a jízdní řád osobní železniční stanice Praha hlavní nádraží [6].

4.1.1 Požadavky

Požadavky na simulační model jsou následující:

- zachycení procesů souvisejících se zpracováním vlaku ve stanici a
- vytvořit podporu pro animaci procesů.

4.1.2 Zjednodušující předpoklady

Protože modelování veškerých procesů souvisejících se zpracováním vlaku je příliš rozsáhlé, byly stanoveny zjednodušující předpoklady:

- simulována bude pouze osobní doprava,
- obsluha vlaku po příjezdu k nástupišti je zjednodušena pouze na naplánování času odjezdu, případně provedení základních operací (přečíslování vlaku, rozpojení vlaku, spojení vlaků),
- nedochází k posunům, výměně lokomotiv apod.,
- je zakázaná úvrať,
- vlak jede po postavené cestě konstantní rychlostí,
- vlak je sledován pouze jako souprava (označená číslem vlaku) nikoliv na úrovni lokomotivy a jednotlivých vagónů,
- postavená cesta pro vlak je uvolněna najednou a
- čas pro postavení cesty je konstantní.

4.2 Výběr programovacího jazyka a vývojového prostředí

Výběr vhodného programovacího jazyka je důležitou součástí vývoje aplikace. Při výběru bylo nutné zvážit požadavky kladené na implementaci výpočetního jádra a simulačního modelu osobní železniční stanice. Jediným požadavkem týkající se programovacího jazyka byl objektově orientovaný programovací jazyk. V úvahy byly vzaty dva programovací jazyky: Java a C#.

V tab. 4.1 je uvedeno základní porovnání programovacích jazyků Java a C#.

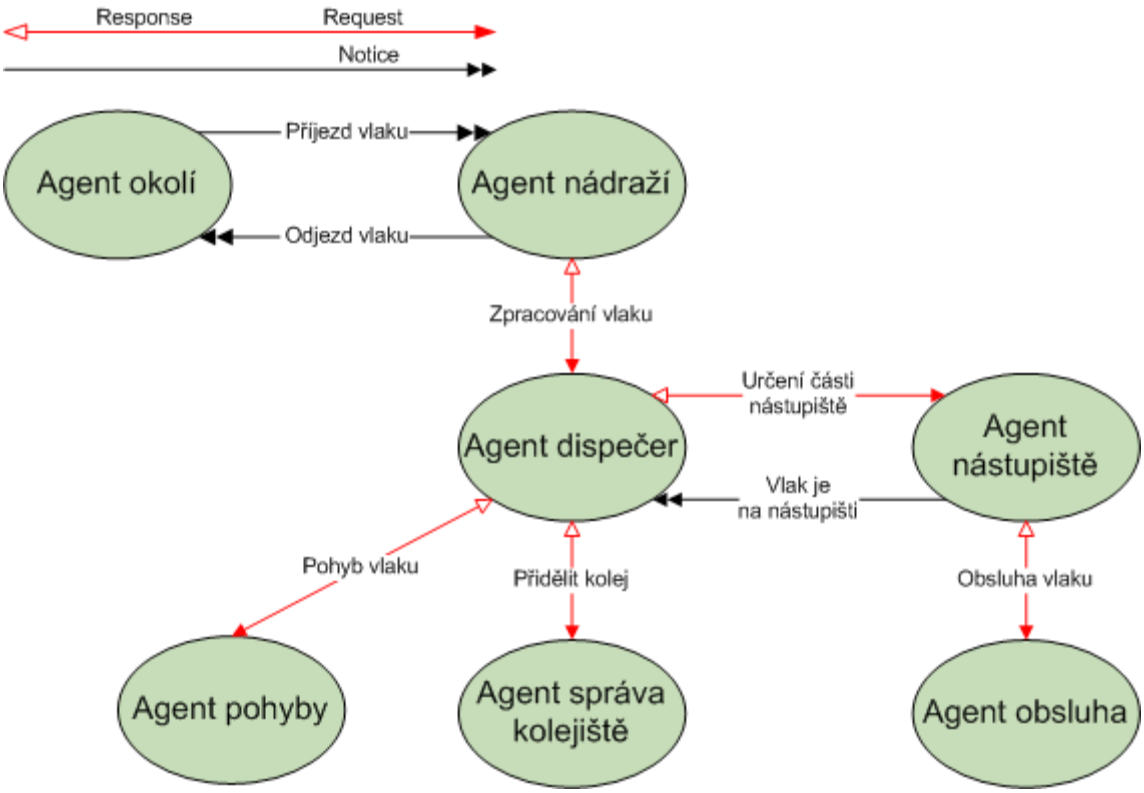
Z tabulky vyplývá, že oba programovací jazyky se v základním porovnání liší pouze v multiplatformnosti. Protože multiplatformnost nebyla požadována, zvolen byl jazyk C#. Jako vývojové prostředí bylo použito Microsoft Visual Studio 2008.

	Java	**C#**
Čistě objektový programovací jazyk	ano	ano
Genericita	ano	ano
Pokročilé vývojové prostředí	ano	ano
Multiplatformnost	ano	ne

Tab. 4.1: Základní porovnání programovacích jazyků (Zdroj: vlastní)

4.3 Konceptuální model

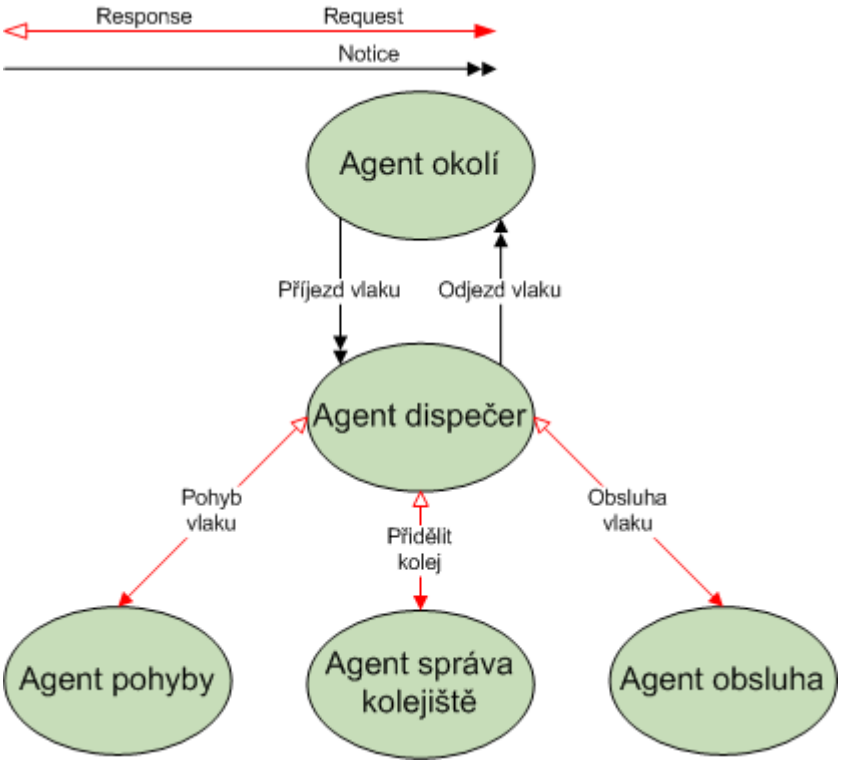
Pro realizaci byla zvolena ABAsim architektura a při formalizaci konceptuálního modelu lze použít její vyjadřovací prostředky. První z koncepcí simulačního modelu (obr. 4.1) je postavena na sedmi agentech: *agent okolí*, *agent nádraží*, *agent dispečer*, *agent pohyby*, *agent správa kolejiště*, *agent nástupiště* a *agent obsluha*. Toto členění se ukázalo jako složité a někteří agenti (agent nádraží a agent nástupiště) se stali nepotřebnými.



Obr. 4.1: První konceptuální model (Zdroj: vlastní)

První koncept byl přepracován a výsledkem byl konceptuální model složený z následujících pěti agentů (obr. 4.2):

- *agenta okolí*, který zahrnuje okolí železniční stanice a zabezpečuje vjezd a odjezd vlaků do/ze systému,
- *agenta dispečera*, který je hlavním agentem pro řízení celé stanice, rozhoduje o pořadí prováděných operací (např. přidělení koleje, pohyb vlaku, vykonání obsluhy),
- *agenta pohyby*, který zabezpečuje pohyb vlaku po kolejišti,
- *agenta správa kolejiště*, který se stará o úkony spojené s kolejištěm (např. postavení cesty, uvolnění cesty, rezervace koleje apod.) a
- *agenta obsluha*, který provádí obsluhu vlaku u nástupiště (naplánování odjezdu, přečíslování, spojení nebo rozpojení vlaku apod.).



Obr. 4.2: Druhý konceptuální model (Zdroj: vlastní)

Konceptuální model tedy naznačuje základní filozofii činnosti simulačního modelu, jehož „životní cyklus“ lze popsat následovně:

- Agent okolí vygeneruje vlak a o jeho příjezdu informuje agenta dispečera.
- Agent dispečer pošle požadavek agentovi správa kolejiště o přidělení (zarezervování) nástupištní koleje.
- Po úspěšném přidělení nástupištní koleje agent dispečer pošle požadavek agentovi pohyby o přemístění vlaku k nástupišti.
- Dále agent dispečer pošle požadavek na obsluhu vlaku agentovi obsluha.
- Po dokončení obsluhy opět agent dispečer pošle žádost o přemístění vlaku od nástupiště agentovi pohyby.
- Nakonec agent dispečer informuje agenta okolí o odjezdu vlaku.

Tento koncepční návrh posloužil jako základ pro vytvoření simulačního modelu, kde byly zpřesněny komunikační vazby.

4.4 Vstupní data

Vstupní data simulačního modelu jsou infrastruktura železniční stanice a transformovaný jízdní řád [6] pro časový interval jednoho týdne bez omezení vlaků na různé dny v roce (např. vlak má definovány výjimečné dny kdy (ne)jede). Získané podklady nebyly v požadovaném formátu, a proto je bylo nutné transformovat do formátu, který lze použít jako vstup simulačního modelu. Byl zvolen formát XML (**E**xtensible **M**arkup **L**anguage). Důvodem tohoto rozhodnutí byla univerzálnost uvedeného formátu a jednoduchá přenositelnost mezi aplikacemi.

4.4.1 Infrastruktura

Datový soubor s definicí infrastruktury obsahuje data o kolejišti (např. koleje, výhybky) a další údaje související s infrastrukturou (definice příjezdových/odjezdových směrů, souvislostí mezi kolejemi). V elementu `<infrastruktura/>` je definováno pět logických bloků obsahující definici kolejí, výhybek, souvislostí, seznamu nástupišť a směrů.

I. První blok definuje seznam jednotlivých kolejí a je označen elementem `<koleje/>`. Obsahem tohoto elementu je seznam jednotlivých kolejí definovaných pomocí elementu `<kolej/>`, uvnitř kterého jsou zaznamenány vlastnosti dané koleje pomocí čtyřech povinných a čtyřech nepovinných elementů:

   - Povinný element `<spojodkud/>` určuje souřadnice počátečního bodu koleje.
   - Povinný element `<spojkam/>` určuje souřadnice koncového bodu koleje.

- Povinný element `<delka/>` určuje délku koleje v metrech.
- Povinný element `<oznaceni>` definuje označení koleje.
- Nepovinné elementy `<bodOK>` a `<bodKO>` definují pomocné body zastavení ve směru od počátečního ke koncovému, resp. od koncového k počátečnímu bodu koleje. Tyto pomocné body se používají např. u nástupištních kolejí, kdy je potřeba, aby vlak zastavil na konci nástupiště.
- Nepovinné elementy `<pomBod1>` a `<pomBod2>` definují pomocné body pro zakřivení koleje. Oba elementy tvoří dvojici a nelze je použít samostatně.

V následující ukázce kódu je maximalistická definice koleje.

```
<kolej>
  <spojodkud>588:84</spojodkud>
  <spojkam>748:97</spojkam>
  <delka>160</delka>
  <oznaceni>1b</oznaceni>
  <bodOK>132</bodOK>
  <bodKO>150</bodKO>
  <pomBod1>724:82</pomBod1>
  <pomBod2>735:88</pomBod2>
</kolej>
```

II. Druhý blok definuje výhybky a je označen elementem `<vyhybky/>`. Obsahem tohoto elementu je seznam jednotlivých výhybek definovaných pomocí elementu `<vyhybka/>`, uvnitř kterého jsou zaznamenány vlastnosti dané výhybky včetně kolejí, které ji tvoří, tzn. že koleje zde definované se nedefinují v bloku pro definici kolejí. Obsahem elementu `<vyhybka/>` jsou dva povinné elementy:

- Element `<koleje/>`, který obsahuje seznam jednotlivých kolejí tvořících výhybku. Výhybka je definovaná minimálně dvěma (pro křížení) a maximálně čtyřmi kolejemi (pro plnohodnotnou anglickou

výhybku). Každá kolej je definovaná v elementu `<kolej/>`, jehož struktura již byla popsána.

- Element `<oznaceni/>`, obsahuje označení výhybky.

V následující ukázce kódu je minimalistická definice výhybky.

```
<vyhybka>
  <koleje>
    <kolej>
      <spojodkud>164:121</spojodkud>
      <spojkam>194:167</spojkam>
      <delka>54</delka>
      <oznaceni>v3k1</oznaceni>
    </kolej>
    <kolej>
      <spojodkud>150:167</spojodkud>
      <spojkam>210:120</spojkam>
      <delka>76</delka>
      <oznaceni>v3k2</oznaceni>
    </kolej>
  </koleje>
  <oznaceni>v3</oznaceni>
</vyhybka>
```

III. Další blok definuje souvislosti mezi kolejemi (např. u nástupiště jsou definovány dvě koleje s označením *20* a *20b* a tyto dvě koleje spolu logicky souvisejí) a je označen elementem `<souvislosti/>`. Obsahem tohoto elementu je seznam souvislostí, které jsou označeny elementem `<koleje/>`, obsahujícího dva povinné elementy `<oznaceni/>`, definující označení souvisejících kolejí. Následující ukázka kódu znázorňuje definici jedné souvislosti.

```
<koleje>
  <oznaceni>20</oznaceni>
  <oznaceni>20b</oznaceni>
</koleje>
```

IV. Předposlední blok obsahuje definici nástupišť a je označen elementem `<seznamnastupist/>`. Obsahem tohoto elementu je seznam nástupišť, které jsou označeny elementem `<nastupiste/>`, obsahujícího povinný

element <oznaceni/>, který definuje označení nástupiště, element <bodTextu/>, který určuje souřadnice vykreslování označení nástupiště, a libovolný počet (minimálně tři) elementů <bod/>, který definuje souřadnice jednoho bodu polygonu, tvořícího obrys nástupiště. Následující ukázka kódu znázorňuje definici jednoho nástupiště.

```
<nastupiste>
  <oznaceni>N1</oznaceni>
  <bod>455:70</bod>
  <bod>724:70</bod>
  <bod>724:50</bod>
  <bod>454:50</bod>
  <bod>455:70</bod>
  <bodTextu>570:55</bodTextu>
</nastupiste>
```

V. Posledním blokem je blok obsahující definici směrů jízdy s průměrnou rychlostí z příjezdových kolejí na nástupištní koleje a naopak. Tento blok je označen elementem <smery/> a obsahuje seznam směrů, které jsou jednotlivě označeny elementem <smer/>. Element <smer/> obsahuje tři povinné elementy a to dva elementy <oznaceni/> (určují označení kolejí definujících směr jízdy a záleží na jejich pořadí) a element <rychlost/>, který definuje průměrnou rychlost pohybu v kilometrech za hodinu v tomto směru. Následující ukázka kódu znázorňuje definici jednoho směru.

```
<smer>
  <oznaceni>103</oznaceni>
  <oznaceni>20</oznaceni>
  <rychlost>50</rychlost>
</smer>
```

4.4.2 Vlaky

Datový soubor s definicí vlaků obsahuje seznam vlaků a jejich omezení. Omezení vlaku lze chápat jako soubor pravidel pro zacházení s vlakem (např. definování dní, ve kterých vlak jede, definování operací, které jsou s vlakem prováděny apod.). Celý seznam vlaků je uzavřen do elementu <vlaky/> a

jednotlivé vlaky jsou definovány v elementu `<vlak/>`, který obsahuje 6 až 10 elementů v závislosti na parametrech vlaku. Povinné elementy jsou: `<cislovlaku/>`, `<druh/>`, `<kolej/>` a `<delka/>`. Dále do skupiny povinných elementů patří elementy `<prijezd/>`, `<odjezd/>`, `<prijezdovakolej/>` a `<odjezdovakolej/>`, ale tyto elementy mají zvláštní pravidla použití:

- vždy musí být definován alespoň jeden element `<prijezd/>` nebo `<odjezd/>`,
- zároveň s elementem `<prijezd/>`, resp. `<odjezd/>`, musí být definován i element `<prijezdovakolej/>`, resp. `<odjezdovakolej/>`,
- samozřejmě lze definovat elementy `<prijezd/>` a `<odjezd/>` najednou.

Do skupiny čistě nepovinných elementů patří elementy `<nazev/>` a `<omezeni/>`.

V následujícím seznamu jsou uvedeny významy jednotlivých elementů:

- Element `<cislovlaku/>` obsahuje číslo daného vlaku.
- Element `<druh/>` obsahuje druh vlaku (např. R, IC, Os, Sv atd.).
- Element `<prijezd/>`, resp. `<odjezd/>`, obsahuje příjezd, resp. odjezd, vlaku.
- Element `<prijezdovakolej/>`, resp. `<odjezdovakolej/>`, obsahuje označení příjezdové, resp. odjezdové koleje, po které daný vlak přijíždí, resp. odjíždí, do/ze systému.
- Element `<nazev/>` obsahuje název vlaku.
- Element `<delka/>` obsahuje délku vlaku v metrech.

- Posledním elementem je `<omezeni/>`, který definuje pravidla pro zacházení s vlakem a obsahuje seznam těchto pravidel. Každé z pravidel je definováno v elementu `<polozka/>`, který obsahuje elementy:
  - Element <dny/>, který určuje pro které dny pravidlo platí. Pokud element <dny/> není uveden, pravidlo platí pro celý týden. Číslování dnů je v pořadí od pondělí do neděle, takže pondělí je označeno číslem *1*, úterý číslem *2* až neděle číslem *7*.
  - Element <operace/>, který definuje operaci pro daný vlak. Obsahuje dva až tři povinné parametry v závislosti na typu operace. Element <typ/> obsahuje jeden ze tří typů operací prováděných s vlakem a to *přečíslování* (`precislovani`), *rozpojení* (`rozpojeni`) a *spojení* (`spojeni`). Další dva elementy `<prvni/>` a `<druhý/>` určují čísla souvisejících vlaků s danou operací. Při operaci *přečíslování* je v elementu `<prvni/>` číslo vlaku, na který se přečíslovává, a element `<druhy/>` je vynechán. Při operaci *rozpojení* jsou v elementech `<prvni/>` a `<druhy/>` uvedena čísla vlaků, na která se původní vlak rozpojuje, přičemž jedno z čísel může být i číslo původního vlaku (část vlaku odjíždí pod původním a část pod novým číslem). Při operaci *spojení* element `<prvni/>` obsahuje číslo vlaku, se kterým se vlak spojuje a element `<druhy/>` obsahuje číslo vlaku po spojení. Pokud element `<operace/>` není uveden, daný vlak jede pouze ve dny uvedené v elementu `<dny/>`, ale není s ním prováděná žádná operace.

Následující ukázka kódu znázorňuje definici jednoho vlaku s omezeními. Je to pouze ilustrační příklad a nemá praktické využití, ale ukazuje všechny možnosti definice vlaku.

```
<vlak>
  <cislovlaku>9120</cislovlaku>
  <druh>Os</druh>
  <prijezd>08:00:00</prijezd>
  <odjezd>08:15:00</odjezd>
  <kolej>20</kolej>
  <prijezdovakolej>301</prijezdovakolej>
  <odjezdovakolej>105</odjezdovakolej>
  <nazev>UKÁZKOVÝ VLAK</nazev>
  <delka>50</delka>
  <omezeni>
    <polozka>
      <dny>1,2</dny>
      <operace>
        <typ>precislovani</typ>
        <prvni>9130</prvni>
      </operace>
    </polozka>
    <polozka>
      <dny>3,4</dny>
      <operace>
        <typ>rozpojeni</typ>
        <prvni>9120</prvni>
        <druhy>9140</druhy>
      </operace>
    </polozka>
    <polozka>
      <dny>5</dny>
      <operace>
        <typ>spojeni</typ>
        <prvni>9150</prvni>
        <druhy>9160</druhy>
      </operace>
    </polozka>
    <polozka>
      <dny>6,7</dny>
    </polozka>
  </omezeni>
</vlak>
```

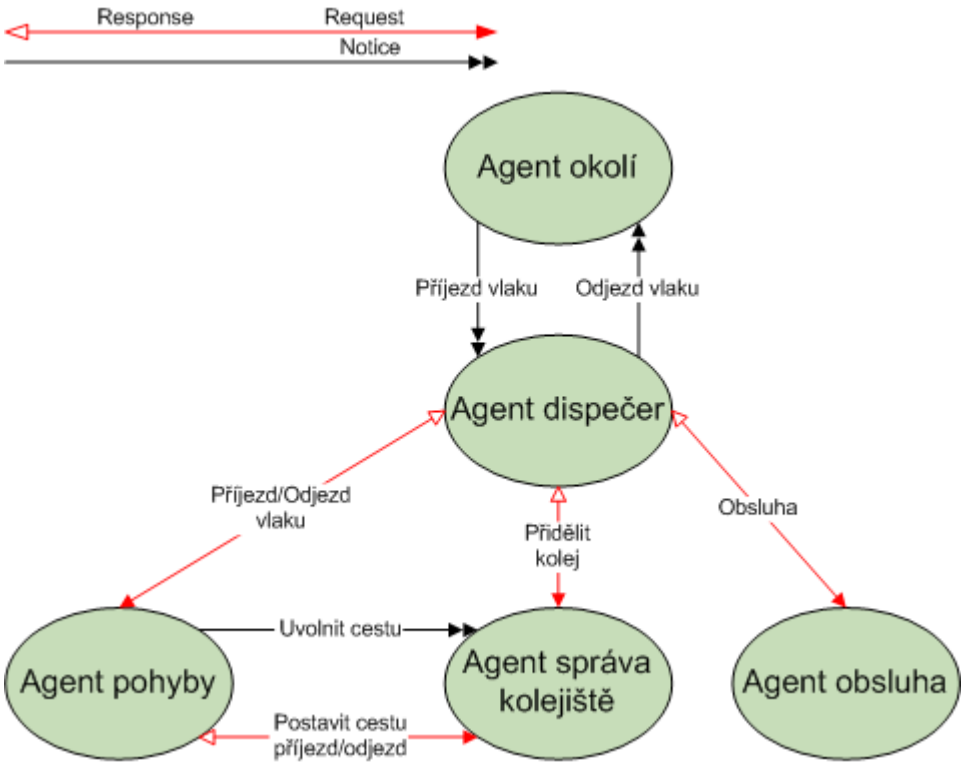
Předchozí ukázkový kód lze přečíst následujícím způsobem:

- Vlak číslo *9120* má každý den v týdnu definovaný příjezd v 8.00 hodin, přijíždí z koleje *301* a zastaví na koleji *20*.
- V pondělí a úterý je vlak *9120* přečíslován na vlak *9130*.
- Ve středu a čtvrtek je vlak rozpojen na vlak *9120* a *9140*, tzn. že část vlaku odjíždí pod původním číslem *9120* a část pod novým číslem *9140*.

- V pátek se vlak *9120* spojuje s vlakem *9150* a po spojení bude nový vlak označen číslem *9160*.
- V sobotu a neděli vlak *9120* odjíždí v 8.15 hodin na odjezdovou kolej *105*.

4.5 Simulační model

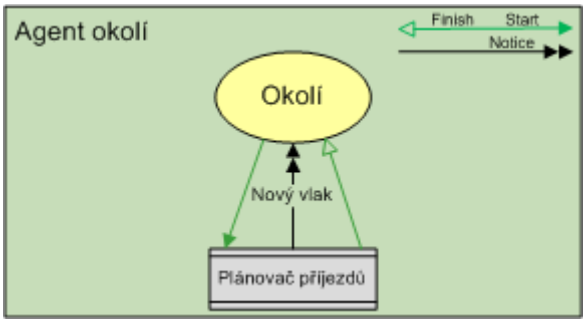
Simulační model vychází z výše uvedeného konceptuálního modelu. Počet i názvy agentů byly zachovány, ale zpřesnily se komunikační vazby mezi agenty. Celý model se tedy skládá z pěti agentů: *agent okolí*, *agent dispečer*, *agent správy kolejiště*, *agent pohyby* a *agent obsluha*. Struktura simulačního modelu je uvedena na obr. 4.3.



Obr. 4.3: Simulační model (Zdroj: vlastní)

4.5.1 Agent okolí

Agent okolí tvoří rozhraní mezi celým systémem a okolním prostředím (další železniční stanice). Na obr. 4.4 je znázorněna vnitřní struktura agenta okolí.



Obr. 4.4: Vnitřní struktura agenta okolí (Zdroj: vlastní)

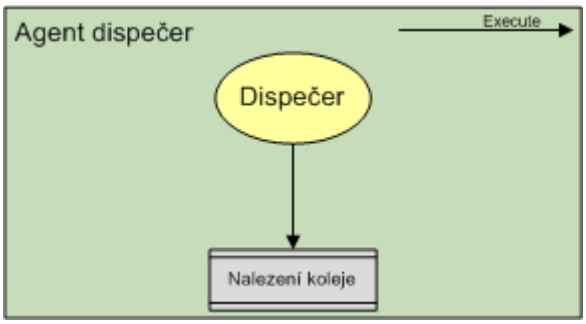
Hlavní funkcí tohoto agenta je plánování příjezdů vlaků do systému a nastavení nezbytných parametrů při odjezdu vlaku ze systému. Plánování je prováděno na základě transformovaného plánu obsazení kolejí s možností generování zpoždění příjezdu vlaku. Generátor zpoždění není součástí implementace této diplomové práce, nicméně výpočetní jádro zajišťuje nezbytnou podporu pro připojení komponentu, který tuto funkčnost realizuje.

Ze znázornění vnitřní struktury (obr. 4.4) je patrné, že agenta okolí tvoří dva komponenty: manažer *okolí* a asistent (proces) *plánovač příjezdů*. Manažer zajišťuje komunikaci s okolním prostředím (ostatními agenty, resp. s agentem dispečerem) a startuje činnost svého asistenta *plánovače příjezdů*. Ten plánuje příjezdy vlaků (řadí vlaky podle času příjezdu na hranici systému do fronty přijíždějících vlaků) a při příjezdu nového vlaku tento vlak zaregistruje do systému a informuje o tom svého manažera.

Při odjezdu vlaku manažer daný vlak ze systému odebere a je mu automaticky systémem nastaven nový čas příjezdu.

4.5.2 Agent dispečer

Agent dispečer je ústřední agent pro řízení dopravy v simulačním modelu železniční stanice a v jeho kompetenci je rozhodování o přijetí či nepřijetí vlaku při příjezdu, odmítnutý vlak potom čeká na na hranici systému do té doby, než mu agent dispečer povolí příjezd. Agent dispečer dále prostřednictvím spolupráce s ostatními agenty řídí provádění technologických postupů (např. přidělení koleje, přemísťování vlaku, obsluha apod.). Na obrázku obr. 4.5 je znázorněna vnitřní struktura agenta dispečera.



Obr. 4.5: Vnitřní struktura agenta dispečera (Zdroj: vlastní)

V kompetenci agenta dispečera je tedy dodržení správného pořadí prováděných operací (komunikace s ostatními agenty a odpovídající reakce na výsledky požadavků) a dále odpovědnost za nalezení vhodné koleje pro vlak, jehož plánovaná kolej je obsazena. Realizace komponentu, zodpovědného za automatické nalezení vhodné koleje, není součástí implementace této diplomové práce, ale výpočetní jádro zajišťuje nezbytnou podporu pro připojení tohoto komponentu.

Vnitřní struktura agenta dispečera (obr. 4.5) je tvořena dvěma komponenty: manažer *dispečer* a asistent (akce) *nalezení koleje*.

Manažer zajišťuje komunikaci s okolím prostředím (resp. s agenty správa kolejiště, pohyby, obsluha a okolí) a iniciuje činnost svého asistenta *nalezení koleje*.

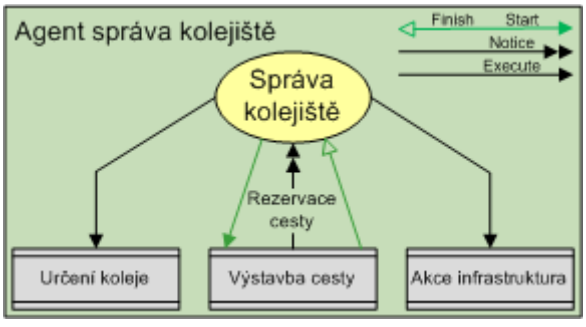
Úkolem asistenta *nalezení koleje* je odpovědět na požadavek manažera o nalezení vhodné koleje, když je kolej, určená pro přijíždějící vlak, obsazena. K tomuto účely byl implementován systém pro výběr komponentu realizujícího rozhodovací mechanizmus.

Pro potřeby testování a ukázky funkčnosti simulačního nástroje železniční stanice [5] byl implementován modul pro interakci s uživatelem (v grafickém režimu). Uživateli jsou zobrazeny informace o vlaku, pro který se nepodařilo kolej přidělit, a seznam všech dostupných kolejí. Po potvrzení výběru koleje pokračuje zpracování asistentem *nalezení koleje*.

Po získání nové koleje z komponentu realizujícího nalezení vhodné koleje je o výsledku informován manažer. Odpovědnost za nalezení vhodné koleje by mohl převzít do své kompetence i agent správa kolejiště, ale při vyhledávání nedochází k žádné změně parametrů prvků infrastruktury, jenom ke zpřístupňování informací, takže tuto odpovědnost lze ponechat v kompetenci agenta dispečera.

4.5.3 Agent správa kolejiště

Agent správa kolejiště se stará o úkony související s infrastrukturou (např. obsazování/uvolňování kolejí, výstavba/rušení cesty apod.). Na obr. 4.6 je znázorněna vnitřní struktura agenta správa kolejiště.



Obr. 4.6: Vnitřní struktura agenta správa kolejiště (Zdroj: vlastní)

V kompetenci agenta správa kolejiště je správa infrastruktury a s tím související operace. Pouze tento agent (s výjimkou agenta obsluha ve speciálních případech) může měnit parametry infrastruktury.

Ze znázornění vnitřní struktury (obr. 4.6) vyplývá, že agenta správa kolejiště tvoří čtyři komponenty: manažer *správa kolejiště* a asistenti (akce) *určení koleje*, (akce) *akce infrastruktura* a (proces) *výstavba cesty*.

Manažer komunikuje se svým okolím (resp. agenty dispečerem a pohyby) a iniciuje činnost svých asistentů.

Komponent *nalezení koleje* slouží k ověření zda požadovaná kolej je volná nebo rezervovaná (případně obsazená). Tato informace je potřebná při příjezdu vlaku, kdy se určuje, na kterou nástupištní kolej bude vlak poslán.

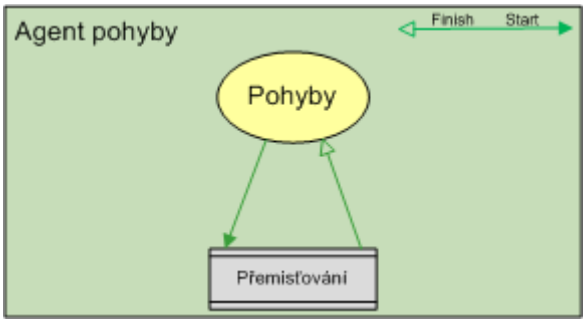
Komponent *akce infrastruktura* má hned několik funkcí: rezervace koleje, uvolnění části postavené cesty a uvolnění celé postavené cesty. Rezervace koleje je opět využívána při příjezdu vlaku, kdy je ověřena nástupištní kolej a je ji zapotřebí rezervovat pro postavení cesty. Funkce uvolnění části cesty se využívá po příjezdu vlaku k nástupišti, kdy je potřeba některé koleje (ty, na kterých vlak stojí) nechat obsazené, ale zbytek postavené cesty uvolnit. A nakonec funkce uvolnění celé postavené cesty je využívána po odjezdu vlaku ze systému, kdy není potřeba nechávat některé koleje obsazené a proto je celá postavená cesta uvolněna.

Posledním komponentem, tvořícího agenta správa kolejiště, je *výstavba cesty*. Jeho úkolem je výstavba cesty pro příjezd vlaku k nástupišti z hranice systému (příjezdová kolej), resp. odjezd vlaku od nástupiště na hranici systému (odjezdová kolej). Tyto dvě funkce se liší v tom, že při výstavbě příjezdové cesty se nemusí brát v úvahu obsazené koleje, na kterých vlak stojí, ale při výstavbě odjezdové cesty je nutné obsazené koleje do cesty započítat. Hlavním důvodem

pro rozlišení je různý způsob výpočtu vzdálenosti přemístění (v metrech) po postavené cestě. To je vzdálenost, kterou ujede vlak od jeho začátku (začátek lokomotivy) po konec cesty, resp. bod zastavení. Takže vzdálenost přemístění může být menší nebo rovna délce postavené cesty (součet délek všech kolejí tvořících postavenou cestu).

4.5.4 Agent pohyby

Agent pohyby se stará o pohyb vlaku po infrastruktuře (např. příjezd, odjezd). Na obr. 4.7 je znázorněna vnitřní struktura agenta pohyby.



Obr. 4.7: Vnitřní struktura agenta pohyby (Zdroj: vlastní)

Hlavními funkcemi agenta pohyby jsou přemístění vlaku při příjezdu (z příjezdové koleje na kolej nástupištní) a při odjezdu (z nástupištní koleje na odjezdovou kolej).

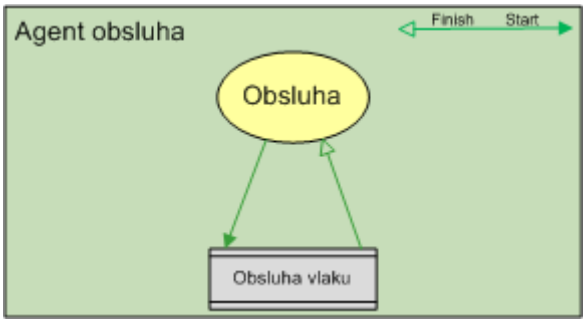
Vnitřní strukturu agenta pohyby (obr. 4.7) tvoří dva komponenty: manažer *pohyby* a asistent (proces) *přemisťování*.

Manažer zajišťuje komunikaci s okolím (resp. s agenty dispečerem a správa kolejiště) a iniciuje činnost svého asistenta.

Komponent *přemisťování* zabezpečuje pohyb vlaku po zadané cestě, resp. naplánuje čas konce pohybu, protože je jedná o diskrétní simulaci. Tento komponent nerozlišuje, zda se jedná o příjezd nebo odjezd vlaku, protože pohyb v obou případech je stejný, liší se pouze v postavené cestě.

4.5.5 Agent obsluha

Agent obsluha se stará o operace související s obsluhou vlaku (např. přečíslování vlaku, spojení vlaků). Na obr. 4.8 je znázorněna vnitřní struktura agenta obsluha.



Obr. 4.8: Vnitřní struktura agenta obsluha (Zdroj: vlastní)

V kompetenci agenta obsluha je provádět operace pro obsloužení vlaku po jeho příjezdu k nástupišti. Při obsluze vlaku mohou nastat 4 případy:

- vlak nemá definovanou žádnou operaci,
- vlak má definovanou operaci přečíslování,
- vlak má definovanou operaci rozpojení,
- vlak má definovanou operaci spojení.

Ze znázornění vnitřní struktury agenta obsluha (obr. 4.8) je patrné, že ho tvoří dva komponenty: manažer *obsluha* a asistent (proces) *obsluha vlaku*.

Manažer zajišťuje komunikaci s okolním prostředím (resp. agentem dispečerem) a iniciuje činnost svého asistenta.

Úkolem komponentu *obsluha vlaku* je provedení výše uvedených případů obsluhy vlaku. Podrobně je lze popsat následovně:

- Vlak nemá definovanou žádnou operaci, a proto je obsluha složena pouze z naplánování času odjezdu od nástupiště, založeného na znalosti skutečného příjezdu vlaku k nástupišti, jeho zpoždění

a předpokládaného času odjezdu. Skutečný čas odjezdu lze stanovit až po příjezdu vlaku k nástupišti.

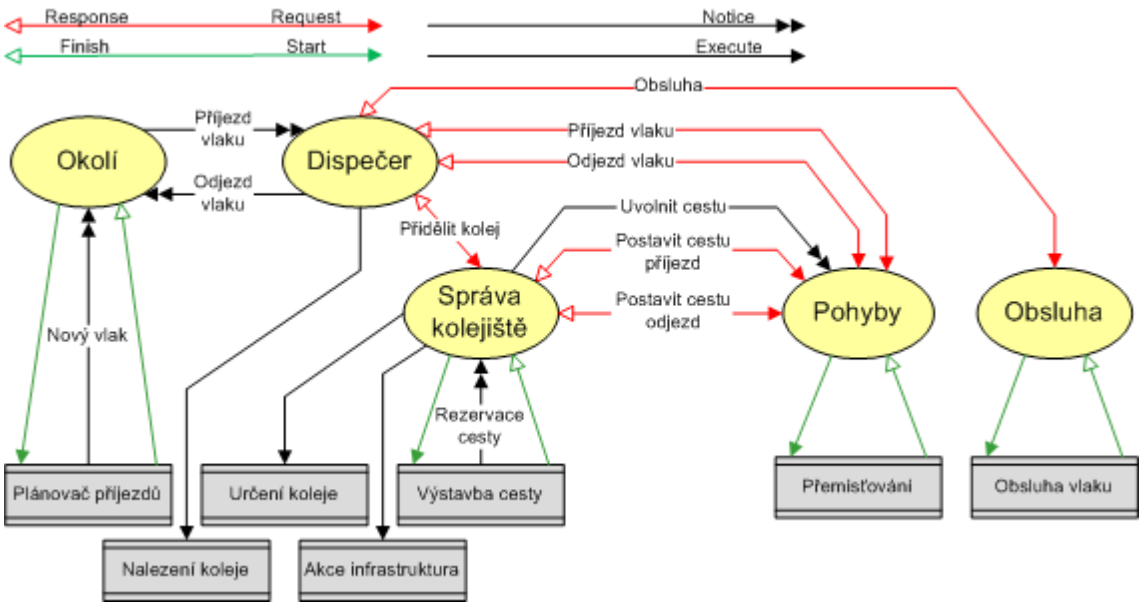
- Vlak s definovanou operací přečíslování je přečíslován a je změněna rezervace a obsazení kolejí, na kterých původní vlak stál (to je jeden ze speciálních případů zásahu do infrastruktury zmíněný v kapitole 4.5.3). Dále je mu naplánován čas odjezdu již popsaným způsobem.
- Vlak s definovanou operací rozpojení se rozpojí na dva vlaky, přičemž jeden vlak může být označen původním číslem vlaku, je změněna rezervace a obsazení kolejí pro oba nové vlaky (to je jeden ze speciálních případů zásahu do infrastruktury zmíněný v kapitole 4.5.3) a oběma vlakům je naplánován čas odjezdu již popsaným způsobem.
- Vlak s definovanou operací spojení může být obsloužen dvěma způsoby v závislosti na pořadí, v kterém přijíždí související vlak pro spojení:
  - Přijíždějící vlak je prvním vlakem pro spojení (u nástupiště nestojí související vlak), proto obsluha nemůže proběhnout a je odložena do doby příjezdu druhého vlaku.
  - Přijíždějící vlak je druhým vlakem pro spojení (u nástupiště již stojí související vlak) a obsluha může proběhnout. Oba vlaky jsou spojeny v jeden, je změněna rezervace a obsazení kolejí pro nový vlak (to je jeden ze speciálních případů zásahu do infrastruktury zmíněný v kapitole 4.5.3) a je mu naplánován čas odjezdu již popsaným způsobem.

Jiné operace se v komponentu *obsluha vlaku* neprovádí, protože ostatní technologické postupy (např. kontrola brzd, přepřah lokomotivy) nejsou v požadavcích kladených na implementaci simulačního modelu osobní železniční stanice.

4.6 Komunikace

V ABAsim architektuře je komunikace realizována výhradně pomocí zasílání zpráv. Každá zpráva musí mít definovaný typ a kód a díky tomu lze jednoznačně rozlišit, jak se daná zpráva má zpracovat. Při implementaci simulačního modelu osobní železniční stanice se postupným vývojem utvořil seznam kódů zpráv, které se v modelu mohou vyskytovat. Dále pro podporu animace bylo nutné vytvořit typy, resp. kódy animačních zpráv.

Na obr. 4.9 je znázorněný zjednodušený vrstvový MPE/ABAsim model osobní železniční stanice. V tomto modelu nejsou zachyceny přesné kódy zpráv, ale pouze zjednodušená komunikace mezi jednotlivými komponenty. Přesný popis komunikace mezi komponenty je popsán v kapitole 4.7.



Obr. 4.9: Zjednodušený vrstvový MPE/ABAsim model osobní železniční stanice (Zdroj: vlastní)

4.6.1 Zprávy

Při implementace simulačního modelu osobní železniční stanice bylo použito 32 různých kódů zpráv popsaných v tab. 4.2. V tabulce jsou uvedeny kódy zpráv, jejich význam a použití ve smyslu, který komponent odesílá zprávu s daným kódem.

Kód zprávy	**Význam a použití**
`CestaNepostavena`	Cesta nepostavena. Využívá komponent *výstavba cesty*.
`CestaPostavena`	Cesta postavena. Využívají komponenty *výstavba cesty*, *manažer správa kolejiště*.
`CestaUvolnena`	Cesta uvolněna. Využívá komponent *akce infrastruktura*.
`DuplikaceOdpovedi`	Duplikace odpovědi. Speciální systémová zpráva využívaná pro duplikaci očekávané odpovědi. Využívá komponent *obsluha vlaku* a obecný komponent *manažer*.
`Inicializace`	Inicializace komponentu. Využívá komponent *manažer okolí* pro inicializaci komponentu *plánovač příjezdů*.
`KolejNeniVolna`	Kolej není volná. Využívá komponent *určení koleje*.
`KolejNepridelena`	Kolej nepřidělena. Využívá komponent *manažer správa kolejiště*.
`KolejPridelena`	Kolej přidělena. Využívá komponent *manažer správa kolejiště*.
`Konec`	Ukončení činnosti komponentu (procesu). Využívá komponent *plánovač příjezdů*.
`KonecObsluhy`	Konec obsluhy vlaku. Využívají komponenty *obsluha vlaku* a *manažer obsluha*.
`KonecPremistovani`	Konec přemísťování vlaku. Využívá komponent *přemísťování*.
`NalezeniKoleje`	Nalezení nové vhodné koleje pro vlak. Využívají komponenty *manažer dispečer* a *nalezení koleje*.
`NovyVlak`	Nový vlak. Využívají komponenty *plánovač příjezdů* a *manažer okolí*.
`ObsaditCestu`	Obsadit cestu. Využívá komponent *výstavba cesty*.
`Obsluha`	Obsluha vlaku. Využívají komponenty *manažer dispečer*, *manažer obsluha* a *obsluha vlaku*.
`OdjezdVlaku`	Odjezd vlaku od nástupiště nebo ze systému. Využívají komponenty *manažer okolí*, *manažer dispečer* a *manažer pohyby*.
`PostavitCestuOdjezd`	Postavit cestu pro odjezd vlaku od nástupiště na hranici systému. Využívají komponenty *manažer pohyby* a *manažer správa kolejiště*.
`PostavitCestuPrijezd`	Postavit cestu pro příjezd vlaku z hranice systému k nástupišti. Využívají komponenty *manažer pohyby* a *manažer správa kolejiště*.
`Precislovani`	Přečíslování vlaku. Využívá komponent *obsluha vlaku*.

PridelitKolej	Přidělení koleje. Využívá komponent *manažer dispečer*.
PrijezdVlaku	Příjezd vlaku do systému nebo k nástupišti. Využívají komponenty *manažer okolí*, *manažer dispečer* a *manažer pohyby*.
RezervaceCesty	Rezervace cesty. Využívá komponent *výstavba cesty*.
RezervovanaKolej	Rezervovaná kolej. Využívá komponent *akce infrastruktura*.
RezervovatKolej	Rezervovat kolej. Využívá komponent *manažer správa kolejiště*.
Rozpojeni	Rozpojení vlaku. Využívá komponent *obsluha vlaku*.
Spojeni	Spojení vlaků. Využívá komponent *obsluha vlaku*.
UrcitKolej	Určení, zda je kolej volná. Využívá komponenta *manažer správa kolejiště*.
UvolnitCastCesty	Uvolnit část cesty. Využívají komponenty *manažer pohyby* a *manažer správa kolejiště*.
UvolnitCelouCestu	Uvolnit celou cestu. Využívají komponenty *manažer pohyby* a *manažer správa kolejiště*.
VolnaKolej	Volná kolej. Využívá komponent *určení koleje*.
ZacatekPremistovani	Začátek přemísťování vlaku. Využívá komponent *manažer pohyby*.
ZruseniOdpovedi	Zrušení odpovědi. Speciální systémová zpráva využívaná pro zrušení očekávané odpovědi. Využívá komponent *obsluha vlaku* a obecný komponent *manažer*.

Tab. 4.2: Kódy zpráv, jejich význam a použití (Zdroj: vlastní)

4.6.2 Animační zprávy

Dalším druhem zpráv jsou zprávy animační, které slouží pro předání informací potřebných pro animační aktivity. Při implementaci simulačního modelu osobní železniční stanice bylo definováno 9 kódů zpráv uvedených v tab. 4.3.

Kód zprávy	**Význam a použití**
DojezdVlaku	Dojezd vlaku. Ukončení přemisťování vlaku. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer pohyby*.
OdjezdVlaku	Začátek přemisťování vlaku z nástupištní koleje na hranici systému (odjezdová kolej). Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer pohyby*.
PostavenaCesta	Postavena cesta. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer správa kolejiště*.
Precislovani	Přečíslování vlaku. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer obsluha*.

`PrijezdVlaku`	Začátek přemisťování vlaku z hranice systému (příjezdová kolej) na nástupištní kolej. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer pohyby*.
`RezervovanaCesta`	Rezervovaná cesta. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer správa kolejiště*.
`RozpojeniVlaku`	Rozpojení vlaku. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer obsluha*.
`SpojeniVlaku`	Spojení vlaků. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer obsluha*.
`UvolnenaCesta`	Uvolněná cesta. Zprávu s tímto kódem do vyrovnávací paměti vkládá komponent *manažer správa kolejiště*.

Tab. 4.3: Kódy animačních zpráv, jejich význam a použití (Zdroj: vlastní)

4.7 Životní cyklus zpracování vlaku

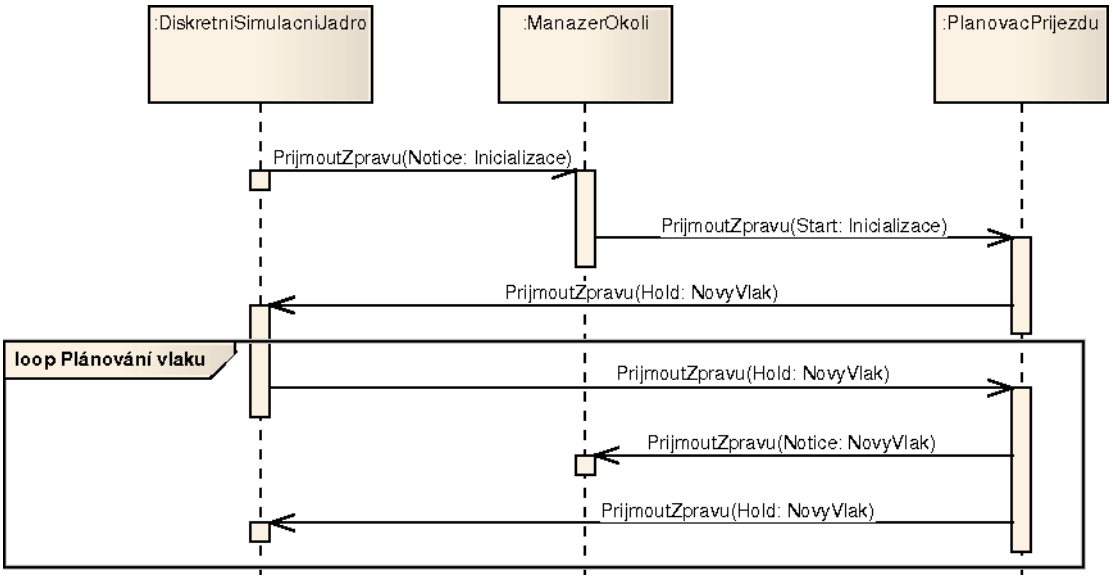
Pro pochopení mechanizmu zpracování vlaku od jeho příjezdu (na hranici systému) až po jeho odjezd (z hranice systému) je vhodné se podívat na jeden jeho cyklus v systému. Pro lepší znázornění a pochopení lze tento cyklus rozdělit na pět fází:

- plánování vlaku,
- přidělení koleje,
- příjezd vlaku,
- obsluha vlaku,
- odjezd vlaku.

Každou z těchto fází lze vysvětlit pomocí komunikačního diagramu znázorňujícího komunikaci mezi komponenty. Na této úrovni nedochází k rozdělení na jednotlivé agenty, ale komunikační diagramy zachycují komplexní komunikaci probíhající mezi všemi komponenty. Hranice vymezující agenty lze stanovit, ale pro zjednodušení je není potřeba zobrazovat, protože vnitřní struktura jednotlivých agentů již byla popsána v kapitole 4.5.

4.7.1 Fáze plánování vlaku

V první fázi plánování vlaku znázorněné na obr. 4.10 je nejprve komponent *plánovač příjezdu* inicializován zprávou typu *Start* od komponentu *manažer okolí*. Inicializace spočívá v naplánování prvního vlaku odesláním zprávy typu *Hold* do centrální pošty diskrétního simulačního jádra. Poté už probíhá cyklus doručení zprávy typu *Hold* z centrální pošty komponentu *plánovač příjezdů*, informování komponentu *manažer okolí* zprávou typu *Notice* o novém vlaku a naplánování dalšího vlaku odesláním zprávy typu *Hold* do centrální pošty. Cyklus je ukončen až ve chvíli ukončení simulace.



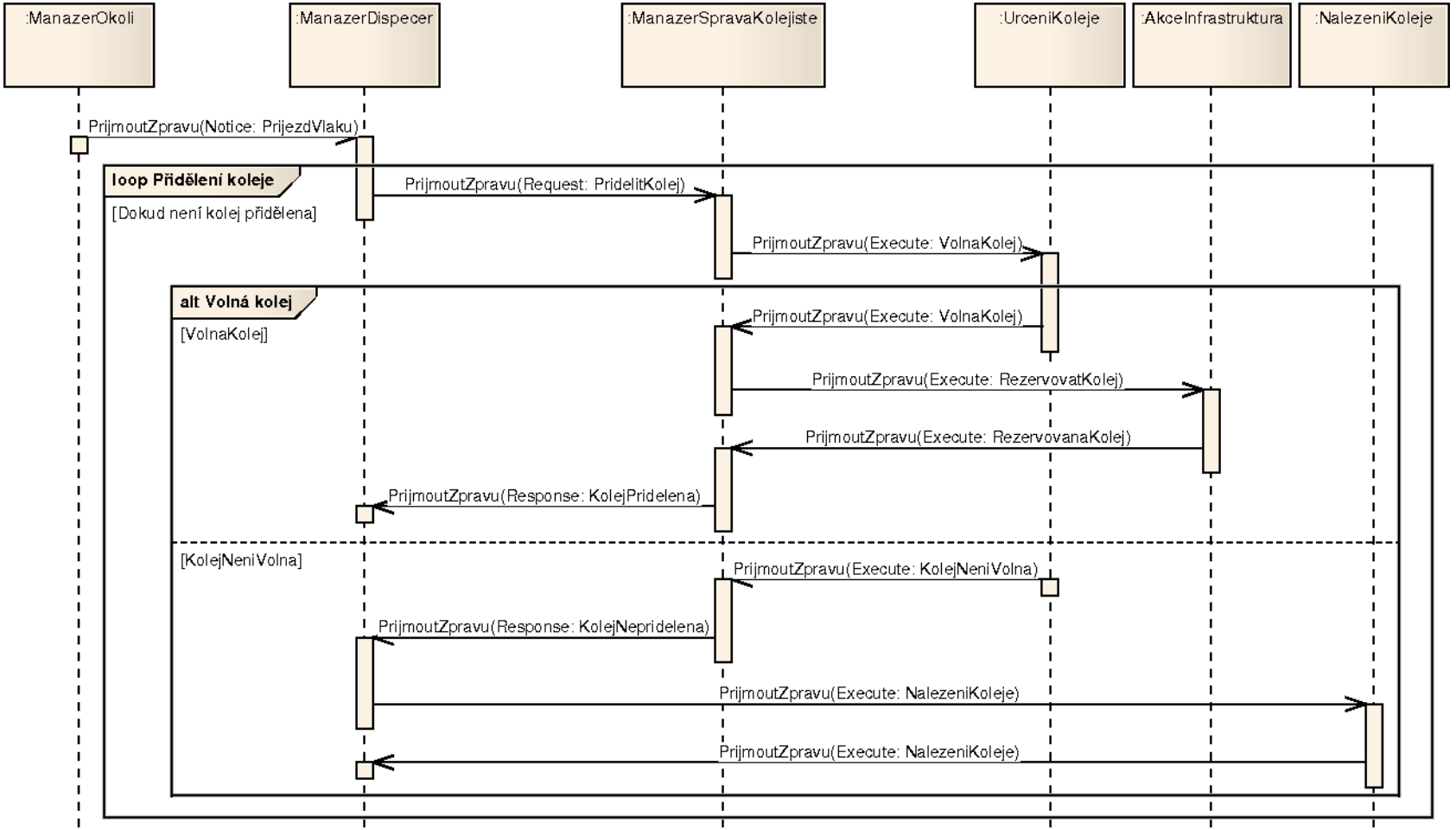
Obr. 4.10: Komunikační diagram plánování vlaku (Zdroj: vlastní)

4.7.2 Fáze přidělení koleje

V druhé fázi přidělení koleje znázorněné na obr. 4.11 nejprve komponent *manažer okolí* pošle komponentu *manažer dispečer* zprávu typu *Notice* o příjezdu vlaku. Ten pošle zprávu komponentu *manažer správa kolejiště* typu *Request* o příjezdu vlaku. Komponent m*anažer správa kolejiště* pošle zprávu typu *Execute* komponentu *určení koleje* s dotazem, zda je kolej volná (nástupištní kolej, na kterou je vlak plánován). Komponent *určení koleje* může odpovědět dvěma způsoby, poté se liší způsob dalšího zpracování:

- Odešle zprávu typu *Execute* s kódem *VolnaKolej* (kolej je buď volná nebo obsazená souvisejícím vlakem) zpět komponentu *manažer správa kolejiště*. Ten pošle zprávu typu *Execute* s požadavkem na rezervování žádané koleje komponentu *akce infrastruktura*. Obratem dostane odpověď (zpráva typu *Execute* odeslaná komponentem *akce infrastruktura*) o zarezervování žádané koleje. Komponent *manažer správa kolejiště* pošle zprávu typu *Response* o přidělení koleje komponentu *manažer dispečer*.
- Odešle zprávu typu *Execute* s kódem *KolejNeniVolna* (kolej je obsazena nebo rezervována pro jiný vlak) zpět komponentu *manažer správa kolejiště*. Ten pošle zprávu typu *Response* zpět komponentu *manažer dispečer*. V tuto chvíli musí komponent *manažer dispečer* poslat zprávu typu *Execute* komponentu *nalezení koleje* s požadavkem na nalezení vhodné nástupištní koleje, který obratem odpoví zprávou typu *Execute*.

V případě, kdy není kolej přidělena, se mechanizmus přidělování opakuje do doby než se podaří vhodnou kolej přidělit.



Obr. 4.11: Komunikační diagram přidělení koleje (Zdroj: vlastní)

4.7.3 Fáze příjezd vlak

Třetí fáze, znázorněná na obr. 4.12, označená jako *příjezd vlaku*, se vyznačuje operacemi souvisejícími s příjezdem vlaku k nástupišti. Konkrétně je to postavení cesty pro příjezd vlaku k nástupišti, přemístění vlaku a uvolnění cesty. Třetí fáze začíná doručením zprávy typu *Notice* komponentu *manažer dispečer* od komponentu *manažer okolí*. Dále komponent *manažer dispečer* pošle zprávu typu *Request* s požadavkem na příjezd vlaku komponentu *manažer pohyby*. Ten pošle požadavek na postavení cesty pro příjezd vlaku pomocí zprávy typu *Notice* komponentu *manažer správa kolejiště*, který pošle zprávu typu *Start* komponentu *výstavba cesty*. Komponent *výstavba cesty* může odpovědět dvěma způsoby, poté se liší způsob dalšího zpracování:

- Odešle zprávu typu *Finish* zpět komponentu *manažer správa kolejiště* a ten uloží původní zprávu s požadavkem na postavení cesty do fronty nepostavených cest.
- Odešle zprávu typu *Notice* zpět komponentu *manažer správa kolejiště* o úspěšné rezervaci cesty a zprávu typu *Hold* do centrální pošty s kódem *ObsaditCestu*. Čas doručení této zprávy zpět komponentu *výstavba cesty* je nastaven tak, aby se simulovala doba potřebná pro fyzické postavení cesty (např. přestavění výhybek, nastavení návěští apod.). Po doručení zprávy zpět komponent *výstavba cesty* odešle zprávu typu *Finish* zpět komponentu *manažer správa kolejiště*. Ten pošle zprávu typu *Response* zpět komponentu *manažer pohyby* s informací, že cesta byla postavena.

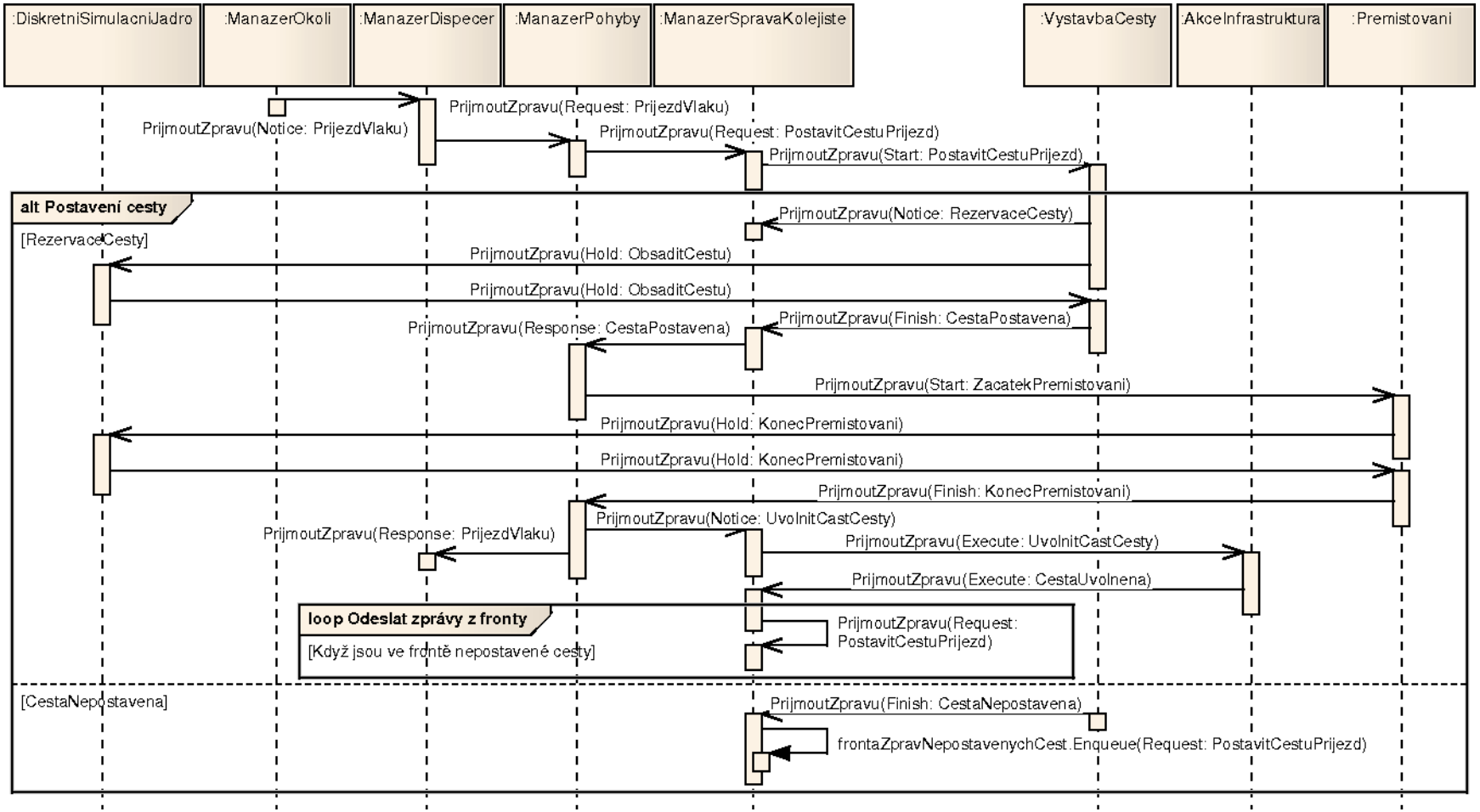
  Následuje odeslání zprávy typu *Start* komponentem *manažer pohyby* komponentu *přemísťování* s požadavkem na přemístění vlaku po postavené cestě. Komponent *přemísťování* naplánuje konec přemísťování pomocí zprávy typu *Hold* odeslané do centrální pošty. Čas doručení této zprávy zpět komponentu *přemísťování* je vypočítán ze vzdálenosti

přemístění po postavené cestě a ze znalosti průměrné rychlosti pohybu ze vstupní koleje na kolej nástupištní. Po doručení zprávy zpět komponentu *přemísťování* tento komponent odešle zprávu typu *Finish* zpět komponentu *manažer pohyby*.

Zbývá pouze upozornit komponent *manažer dispečer* o příjezdu vlaku k nástupišti odesláním zprávy typu *Response* komponentem *manažer pohyby* a uvolnění postavené cesty odesláním zprávy typu *Notice* komponentem *manažer pohyby* komponentu *manažer správa kolejiště*. Ten odešle zprávu typu *Execute* komponentu *akce infrastruktura* s požadavkem na uvolnění části cesty (popsáno v kapitole 4.5.3). Komponent *akce infrastruktura* obratem odpoví komponentu *manažer správa kolejiště* zprávou typu *Execute* o úspěšném uvolnění cesty.

Definitivně poslední operací fáze příjezdu vlaku je vložení všech zpráv z fronty nepostavených cest do vlastní poštovní schránky komponentu *manažer správa kolejiště*. Všechny znovu přijaté zprávy s požadavkem na postavení cesty jsou zpracovávány stejným výše popsaným způsobem. Některé cesty je možné postavit (byly uvolněny některé prvky infrastruktury) a jiné ne, proto jsou opět zařazeny do fronty nepostavených cest.

Příjezd vlaku tedy není triviální fází zpracování a je zde možnost zpoždění vlaku z důvodu čekání na postavení cesty.

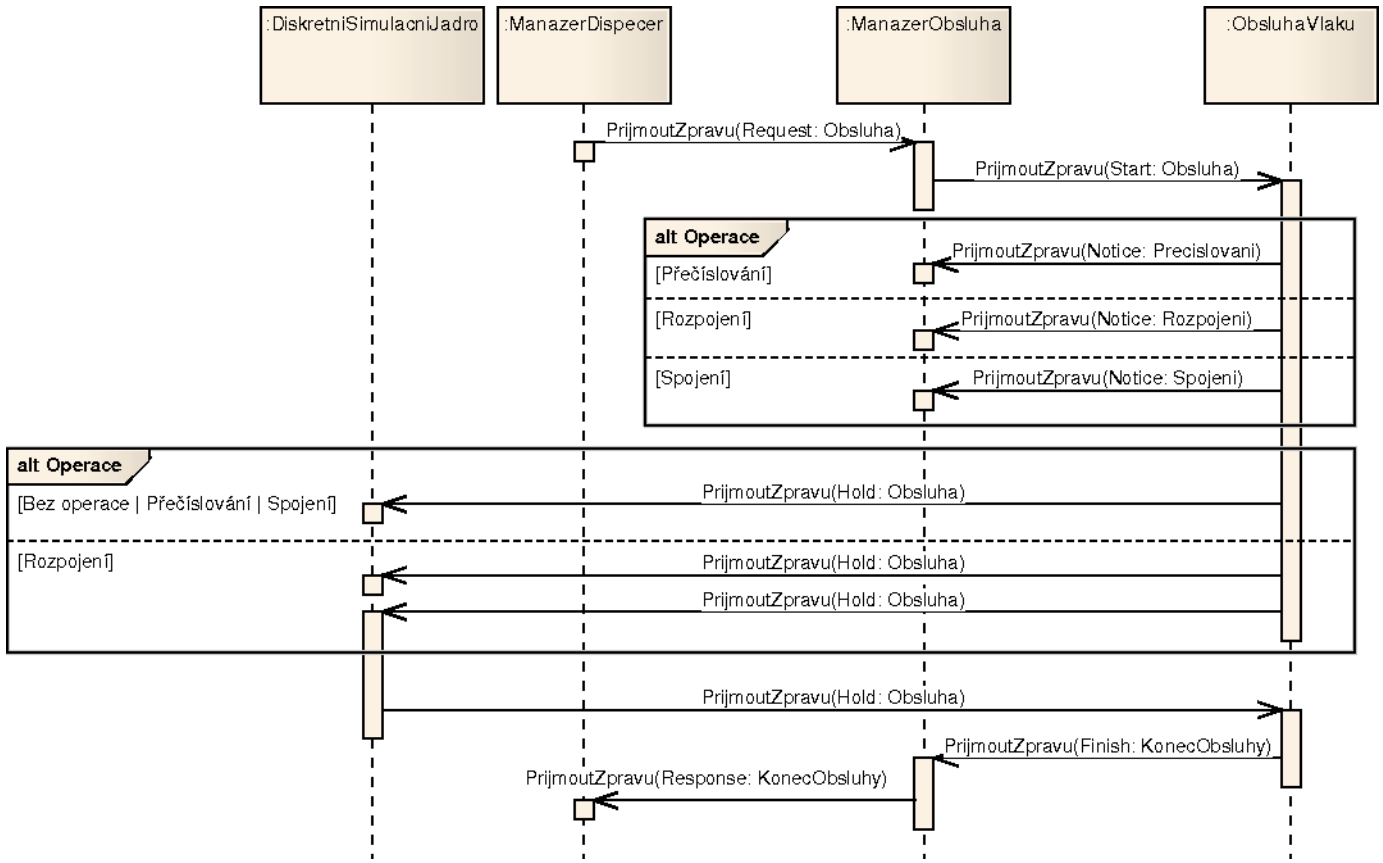


Obr. 4.12: Komunikační diagram příjezdu vlaku (Zdroj: vlastní)

4.7.4 Fáze obsluha vlaku

Ve čtvrté fázi dochází k obsluze vlaku, který přijel k nástupišti. Tato fáze je znázorněna na obr. 4.13. Obsluhou vlaku je myšleno naplánování jeho odjezdu po provedení některé operace (přečíslování, rozpojení vlaku, spojení vlaků). Pokud vlak nemá uvedenou žádnou operaci, je mu odjezd naplánován ihned. Provádění jednotlivých operací bylo popsáno v kapitole 4.5.5, proto v této kapitole bude popis zaměřen na komunikaci komponentů.

Fáze obsluhy vlaku je započata ve chvíli odeslání zprávy typu *Request* komponentem *manažer dispečer* komponentu *manažer obsluha*. Ten odešle zprávu typu *Start* komponentu *obsluha vlaku*. Komponent *obsluha vlaku* provede již zmíněné operace a o jejich provedení informuje zprávou typu *Notice* komponent *manažer obsluha*. Pokud vlak neměl definovanou žádnou operaci, tato zpráva není poslána. Následuje odeslání jedné (v případě, kdy vlak nemá definovanou operaci nebo při operaci přečíslování a spojení) nebo dvou (v případě, kdy je definována operace rozpojení, protože původní vlak je rozpojen na dva nové) zpráv typu *Hold* o dokončení obsluhy v daný čas do centrální pošty. Čas doručení zprávy zpět komponentu *obsluha vlaku* je nastaven podle plánované doby odjezdu daného vlaku s přihlédnutím na jeho zpoždění. Tato doba je navíc zkrácena o dobu postavení cesty pro odjezd vlaku, aby vlak odjížděl od nástupiště v čase, který má naplánovaný. Po doručení zprávy zpět komponentu *obsluha vlaku* tento komponent odešle zprávu typu *Finish* komponentu *manažer obsluha* s informací o konci obsluhy a ten odešle zprávu typu *Response* zpět komponentu *manažer dispečer* s informací o dokončení obsluhy vlaku.



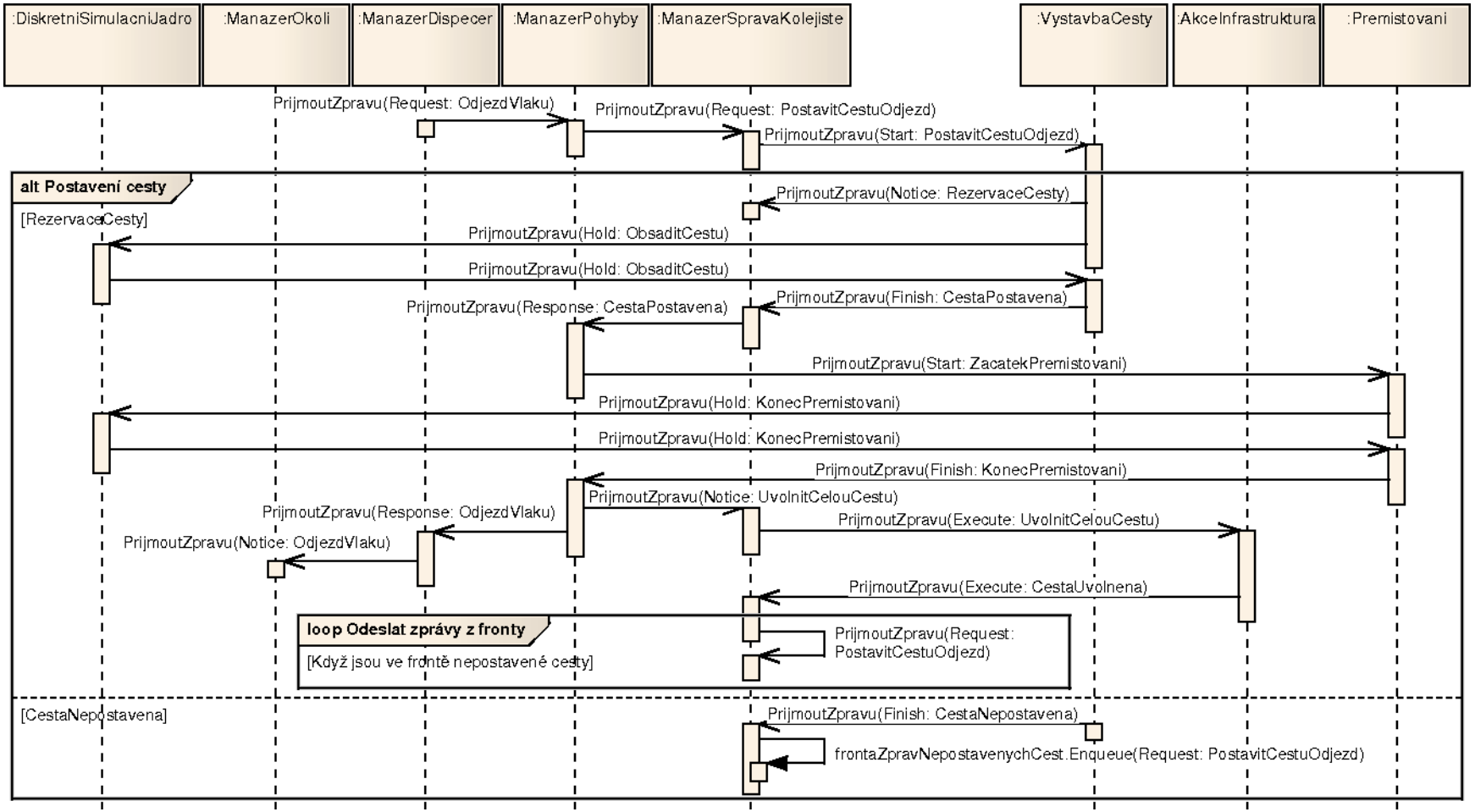
Obr. 4.13: Komunikační diagram obsluhy vlaku (Zdroj: Vlastní)

4.7.5 Fáze odjezd vlaku

V páté a poslední fázi dochází k úkonům spojeným s odjezdem vlaku od nástupiště na hranici systému (odjezdovou kolej). Znázorněná je na obr. 4.14. Tato fáze je takřka shodná s fází příjezdu vlaku popsané v kapitole 4.7.3. Z pohledu komunikace se obě fáze od sebe liší počátkem, kdy fáze odjezdu vlaku není započata zprávou *Notice* od komponentu *manažer okolí* komponentu *manažer dispečer*, a koncem, kdy je ve fázi odjezdu vlaku navíc odeslána zpráva typu *Notice* komponentem *manažer dispečer* komponentu *manažer okolí* s informací o odjezdu vlaku.

Další rozdíl lze najít v kódech odesílaných zpráv (např. kód zprávy *OdjezdVlaku* místo kódu *PrijezdVlaku* nebo *PostavitCestuOdjezd* místo *PostavitCestuPrjezd*) a tím samozřejmě odlišné vnitřní chování komponent. Důležitým rozdílem v kódech odesílaných zpráv je ta chvíle, kdy komponent *manažer pohyby* odesílá zprávu typu *Notice* komponentu *manažer správa kolejiště*. Pří fázi příjezdu vlaku je kód zprávy nastaven na *UvolnitCastCesty*, ale při odjezdu je nastaven na *UvolnitCelouCestu*. Vnitřní chování komponenty *akce infrastruktura* v těchto případech je popsáno v kapitole 4.5.3.

Podobně jako ve fázi příjezdu vlaku i zde může dojít ke zpoždění vlaku z důvodu čekání na postavení cesty pro odjezd.



Obr. 4.14: Komunikační diagram odjezdu vlaku (Zdroj: vlastní)

4.8 Vybraná implementační řešení

V této kapitole a podkapitolách budou popsána některá vybraná implementační řešení. Konkrétně popis implementace datové struktury uchovávající infrastrukturu, algoritmu hledání nejkratší cesty v kolejišti a řídícího algoritmu diskrétní simulace s on-line animací. Při implementaci byly použity pokročilé techniky programování [7][8] (např. generické programování).

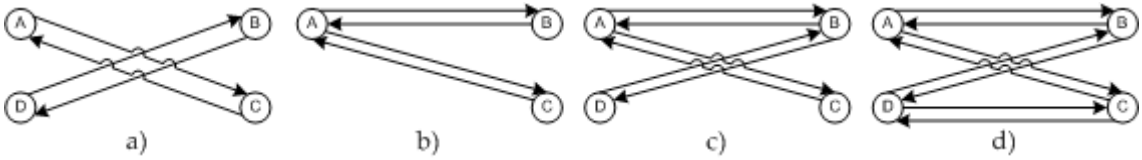
4.8.1 Infrastruktura

Implementace infrastruktury (kolejiště) je rozvržena do dvou vrstev a to z pohledu vnější reprezentace prvků infrastruktury a jejich vnitřní struktury. Z vnějšího pohledu se infrastruktura skládá ze základních stavebních prvků – kolejí. Ty dále tvoří složitější prvky – výhybky. Vnitřní struktura koleje se skládá ze dvou bodů (definujících koncové body koleje, ve kterých se koleje spojují) a dvou orientovaných hran (definujících povolený směr jízdy po dané koleji), znázorněna je na obr. 4.15.

Obr. 4.15: Znázornění vnitřní struktury koleje (Zdroj: vlastní)

Obě hrany koleje jsou ohodnoceny stejnou hodnotou a tou je délka koleje. Důvodem zdvojení hrany je jednodušší použití dané koleje v algoritmu pro vyhledávání nejkratší cesty v kolejišti.

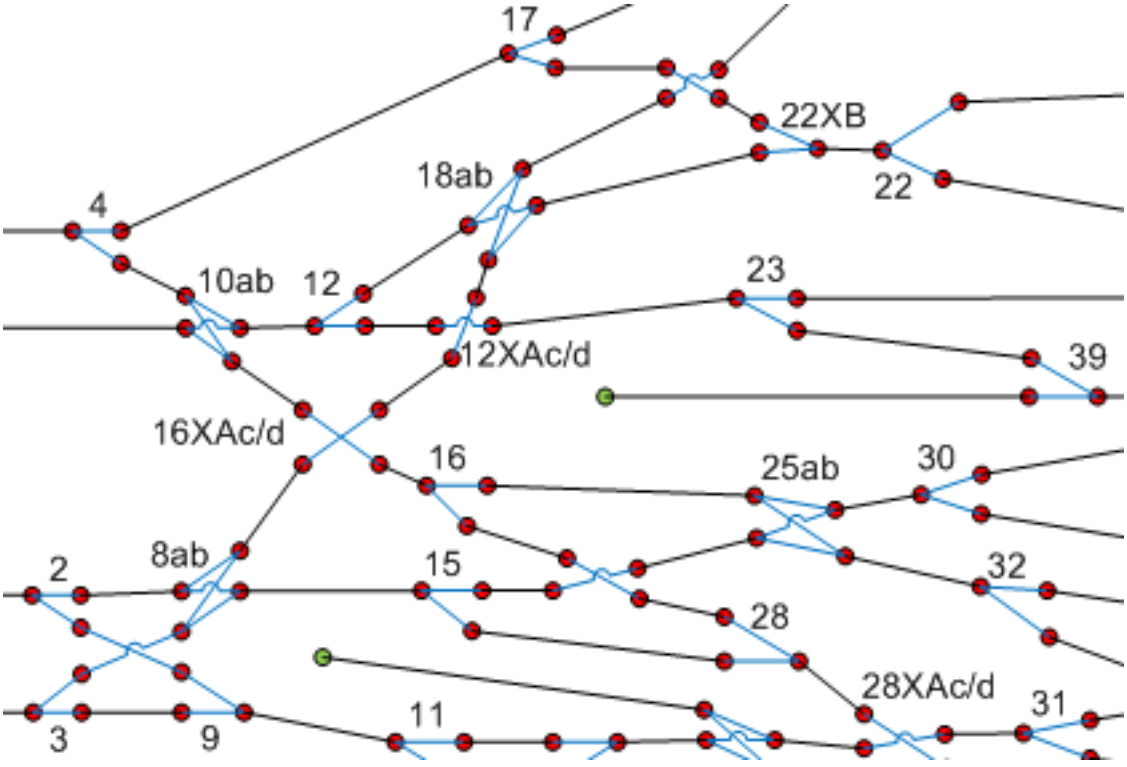
Vnitřní struktura výhybky, případně křížení, se skládá z několika kolejí a to tří nebo čtyř bodů navzájem pospojovaných orientovanými hranami podle typu výhybky, případně křížení. Vnitřní reprezentace jednotlivých typů je znázorněna na obr. 4.16.



Obr. 4.16: Znázornění vnitřní struktury **a)** křížení **b)** jednoduché výhybky **c)** poloviční anglické výhybky **d)** anglické výhybky (Zdroj: vlastní)

Prvky infrastruktury (koleje a výhybky) se dají navzájem spojovat do většího celku tvořícího kolejiště. Spojením jednotlivých prvků vznikne graf s násobnými a orientovanými hranami bez smyček, tedy multidigraf.

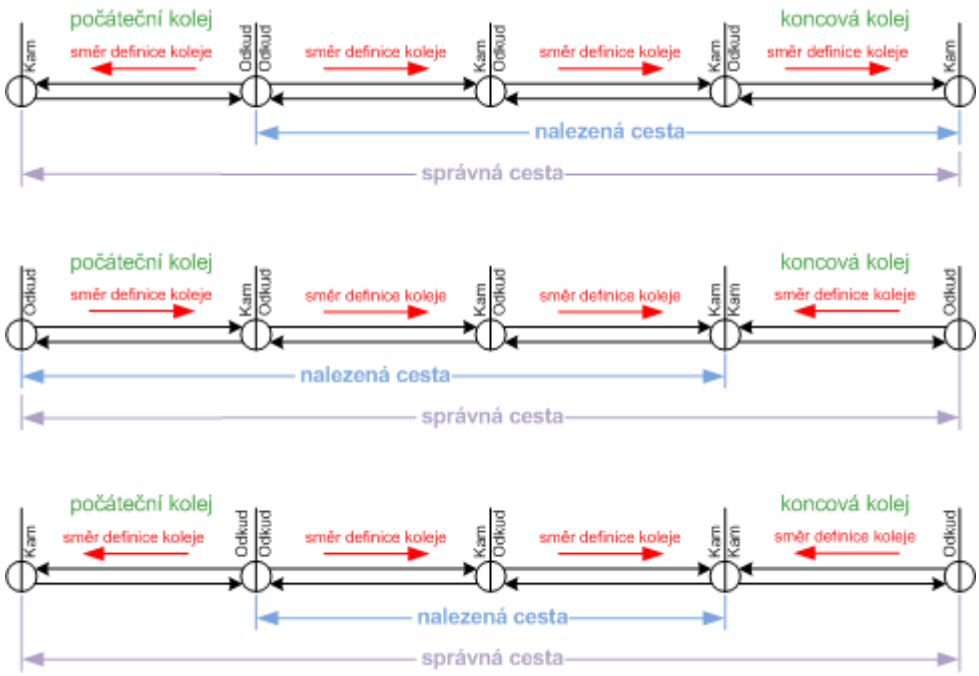
Z výše uvedeného vyplývá, že v první vrstvě je uchováváno kolejiště v podobě grafu a nejsou zde rozlišeny jednotlivé prvky infrastruktury (koleje a výhybky). Druhá vrstva tedy definuje na základě jednotlivých bodů a hran mezi těmito body jednotlivé prvky infrastruktury koleje a výhybky a jejich parametry. Na obr. 4.17 je znázorněna ukázka části zjednodušeného grafu s vyznačením vnitřní struktury prvků infrastruktury.



Obr. 4.17: Ukázka části zjednodušeného grafu s vyznačením vnitřní struktury prvků (Zdroj: vlastní)

4.8.2 Algoritmus hledání nejkratší cesty v kolejišti

Problém hledání nejkratší cesty v kolejišti [9] byl vyřešen použitím upraveného Dijkstrova algoritmu [10]. Vlastností Dijkstrova algoritmu je vyhledání nejkratší cesty mezi dvěma vrcholy, je vrcholově orientovaný. Problém, který bylo potřeba vyřešit, byl ten, že nejkratší cesta se hledá z koleje na kolej (každá kolej je definovaný dvěma body), takže při použití neupraveného Dijkstrova algoritmu mohlo dojít k situaci, kdy byla vynechána počáteční nebo koncová kolej nebo obě dvě. Při definování koleje nezáleží na pořadí bodů, které ji tvoří, ale při hledání cesty je nutné najít nejkratší cestu mezi dvěma kolejemi a zároveň nejdelší mezi jejich body. Na obr. 4.18 jsou znázorněny tři možné případy, kdy je nalezena špatná cesta při použití neupraveného Dijkstrova algoritmu a vstupními daty jsou dva body (bod *Odkud* počáteční a bod *Kam* cílové koleje).



Obr. 4.18: Ukázka špatně nalezených cest (Zdroj: vlastní)

Dále bylo nutné vyřešit zákaz naplánování cesty přes rezervovanou/obsazenou kolej (s výjimkou rezervované/obsazené koleje souvisejícím vlakem) a zakázané odbočení na výhybce.

V následující ukázce je uveden Dijkstrův algoritmus v pseudokódu v neupravené původní verzi.

```
1  function Dijkstra(E, V, s): //E je množina všech hran
                                 V je množina všech vrcholů
                                 s je počáteční vrchol
2    for each vertex v in V:    //Inicializace
3      d[v] := infinity         //Zatím neznámá vzdálenost
                                 z počátku s do vrcholu v
4      p[v] := undefined        //Předchozí vrchol na
                                 nejkratší cestě z počátku s
                                 k cíli
5    d[s] := 0                  //Vzdálenost z s do s
6    N := V                     //Všechny dosud nenavštívené
                                 vrcholy
7    while N is not empty:      //Samotný algoritmus
8      u := extract_min(N)      //Vezměme "nejlepší" vrchol
9      if u = target            //Ukončovací podmínka
10       break
11     for each neighbor v of u:
12       alt = d[u] + l(u, v)  //pozor v 1. smyčce cyklu
                                 d[u] je ještě nekonečno
13       if alt < d[v]
14         d[v] := alt
15         p[v] := u
```

Výslednou cestu z počátku do cíle lze zjistit cyklem:

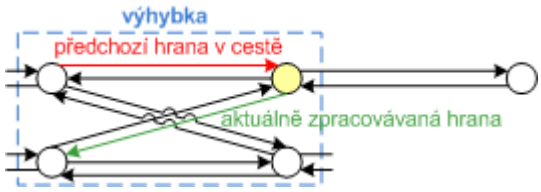
```
1 S := empty sequence
2 u := target
3 while defined p[u]
4   insert u at the beginning of S
5   u := p[u]
```

První úpravou Dijkstrova algoritmu byla možnost jako vstup zadat dvojici kolejí místo dvojice bodů (bod *Odkud* počáteční a bod *Kam* cílové koleje). S tím souvisí změna inicializace, kdy jsou nastaveny vzdálenosti (`d[počátečníKolej.Odkud]` a `d[počátečníKolej.Kam]`) obou počátečních bodů (počáteční a koncový bod počáteční koleje). Touto úpravou dojde k vynechání hrany mezi počátečním a koncovým bodem počáteční koleje. Ta je

do cesty zařazena dodatečně na konci algoritmu. Dále bylo nutné upravit ukončovací podmínku, aby hledání nejkratší cesty skončilo až ve chvíli, kdy byly navštíveny oba koncové body (počáteční a koncový bod cílové koleje).

Další úpravou bylo testování rezervace koleje, aby nedošlo k naplánování cesty, když je kolej zarezervovaná pro jiný vlak kromě vlaku, který má souvislost s vlakem, pro který je cesta hledána. Koleje, které nevyhovují, jsou vynechány a pokračuje se další kolejí.

Definice zakázaného odbočení na výhybce byla do Dijkstrova algoritmu zakomponována tak, že se testuje předchozí hrana aktuálně zpracovávané hrany, zda nepatří koleji tvořící výhybku a ta je shodná s výhybkou, které patří aktuálně zpracovávaná hrana. Když je podmínka splněna, daná hrana je vynechána. Tento případ je znázorněn na obr. 4.19.



Obr. 4.19: Zpracování zakázaného odbočení na výhybce (Zdroj: vlastní)

Poslední úpravou je změna ve výsledném sestavení cesty. Ve skutečnosti jsou sestaveny dvě cesty a jako výsledná je vybrána ta delší (odpovídá hledané cestě včetně počáteční a cílové koleje).

Výsledný upravený Dijkstrův algoritmus v pseudokódu je uveden v následující ukázce:

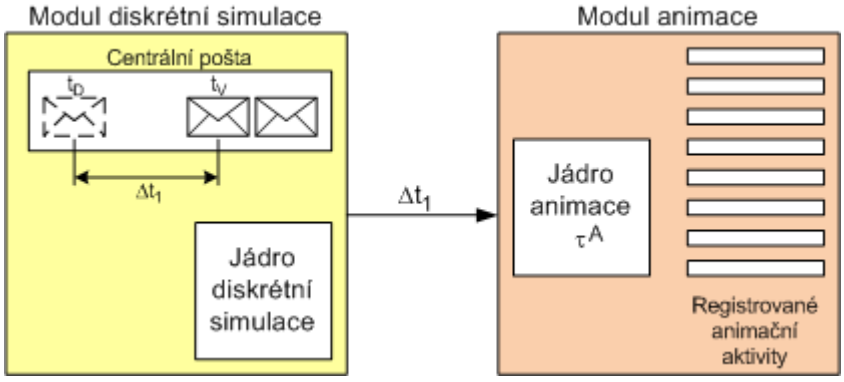
```
1  function Dijkstra(G, o, k)   //G je graf
                                  o je počáteční kolej
                                  k je cílová kolej
2  for each vrchol v in G.V     //G.V všechny vrcholy grafu
3    d[v] := infinity
4    p[v] := undefined
5  d[o.Odkud] := 0
6  d[o.Kam] := 0
7  N := G.V                     //všechny nenavštívené vrcholy
8  while N is not empty:
9    u := extract_min(N)
10   if byly navštíveny oba vrcholy k.Odkud a k.Kam
11     break
12   for each neighbor v of u:
13     e := getEdge(u, v)       //hrana z u do v
14     if e is rezervovaná pro nesouvisející vlak
15       continue
16     if p[u] is defined       //test existence předchozího
                                  vrcholu
17       pe := getEdge(p[u], u) //předchozí hrana
18       vyhyb1 := getVyhybka(e) //získání výhybky, které
                                   patří hrana
19       vyhyb2 := getVyhybka(pe)
20       if vyhyb1 and vyhyb2 is defined and vyhyb1 = vyhyb2
21         continue
22     alt = d[u] + l(u, v)
23     if alt < d[v]
24       d[v] := alt
25       p[v] := u
```

Výsledné sestavení cesty:

```
1   cesta := cesta1 := cesta2 := empty
2   v := k.Odkud
3   pv := k.Odkud              //předchozí vrchol
4   while p[v] is defined
5     v := p[v]
6     vložit na začátek cesta1 hranu (v, pv)
7     pv := v
8   vložit na začátek cesta1 hranu počáteční koleje
9   v := k.Kam
10  pv:= k.Kam
11  while p[v] is defined
12    v := p[v]
13    vložit na začátek cesta2 hranu (v, pv)
14    pv := v
15  vložit na začátek cesta2 hranu počáteční koleje
16  cesta := delší z {cesta1, cesta2}
```

4.8.3 Řídící algoritmus diskrétní simulace s on-line animací

Protože při implementaci výpočetního jádra nebyl požadavek na zpracování spojitých aktivit, bylo vynecháno jádro spojité simulace a tím se změnil i algoritmus řízení. Nyní je tedy potřeba řídit pouze modul diskrétní simulace a modul animace. Na obr. 4.20 je znázorněno přidělování časových kvant mezi modulem diskrétní simulace a animace.



Obr. 4.20: Přidělování časových kvant mezi modulem diskrétní simulace a animace (Zdroj: vlastní)

Rozdíl mezi původním řídícím algoritmem popsaným v kapitole 3.2 a upraveným pro řízení pouze modulu diskrétní simulace a animace uvedeným v tabulce 4.4 je ve vynechání řízení modulu spojité simulace a přidáním jednoho předání řízení modulu animace v kroku 3. Tato úprava je z důvodu situací, kdy nedochází k plánování událostí cyklicky, ale pouze je provedena série úloh, jejichž provádění skončí dříve než je čas ukončení simulace a v animačním modulu jsou zaregistrovány animační aktivity, které ještě nebyly ukončeny. Tímto je tedy dán prostor k dokončení animačních aktivit před ukončením simulace.

Krok	Vykonává	Činnost	Vykonaná za podmínky
0	**Jádro diskrétní simulace**	Inicializace simulačního času diskrétní simulace $\mathbf{t_D}$ ($\mathbf{t_D} = 0$).	
1		Zjištění celkového počtu zpráv $\mathbf{n_P}$ v poštovních schránkách všech komponentů a zjištění naplněnosti kalendáře (centrální pošty). $\mathbf{n_K} = 0$ … prázdný kalendář $\mathbf{n_K} > 0$ … neprázdný kalendář	
2		Zjištění časového kvanta $\mathbf{\Delta t_1}$. Pozn.: tato situace může nastat pouze v případě, kdy nejsou aktivity plánovány cyklicky.	Platí ($\mathbf{n_P} = 0 \wedge \mathbf{n_K} = 0$) a simulační čas diskrétní simulace je menší než čas ukončení ($\mathbf{t_D} < \mathbf{t_U}$).
3	**Jádro animace**	Aplikace metody snímání animačních aktivit se snímací periodou $\boldsymbol{\tau^A}$ až do vyčerpání svého časového kvanta $\mathbf{\Delta t_1}$.	$\mathbf{\Delta t_1} \neq 0$ a počet zaregistrovaných „spojitých“ aktivit $\mathbf{n_C} \neq 0$
4	**Jádro diskrétní simulace**	Ukončení simulace.	Platí ($\mathbf{n_P} = 0 \wedge \mathbf{n_K} = 0$) nebo je vyčerpán určený čas pro běh simulačního programu.
5		Postupné vydávání pokynů všem komponentům k výběru zpráv ze svých poštovních schránek a k jejich okamžitému zpracování. Pozn.: vydávání zmíněných pokynů je ukončeno jestliže $\mathbf{n_P} = 0$.	$\mathbf{n_P} > 0$
6		Zjištění časového kvanta $\mathbf{\Delta t_1}$.	
7	**Jádro animace**	Provedení registrace animačních aktivit na základě údajů z vyrovnávací paměti.	$\mathbf{\Delta t_1} \neq 0$
8		Aplikace metody snímání animačních aktivit se snímací periodou $\boldsymbol{\tau^A}$ až do vyčerpání svého časového kvanta $\mathbf{\Delta t_1}$.	$\mathbf{\Delta t_1} \neq 0$ a $\mathbf{n_C} \neq 0$
9	**Jádro diskrétní simulace**	Aktualizuje simulačního času diskrétní simulace $\mathbf{t_D} = \mathbf{t_V}$, kde $\mathbf{t_V}$ představuje hodnotu nejmenšího časového razítka ze všech zpráv aktuálně obsažených v kalendáři (centrální poště).	$\mathbf{n_K} > 0$
10		Výběr všech zpráv z kalendáře (centrální pošty) s časovým razítkem $\mathbf{t_V}$ a jejich doručení do poštovních schránek příslušných adresátů.	$\mathbf{n_K} > 0$
11		Návrat na **KROK 1**.	

Tab. 4.4: Řídící algoritmus diskrétní simulace s on-line animací (Zdroj: vlastní)

4.9 Možnosti rozšíření

Možnosti rozšíření výpočetního jádra a simulačního modelu bez zásahu do jejich implementace jsou následující:

- připojení komponentu (i více najednou), který implementuje nalezení vhodné nástupištní koleje pro vlak, jehož plánovaná kolej je obsazena,
- připojení komponentu, který implementuje generátor zpozdění.

Možným rozšířením výpočetního jádra je implementace řídícího algoritmu zahrnujícího podporu modulu spojité simulace. Modul spojité simulace nebyl při implementaci této diplomové práce potřeba, a proto nebyla implementována podpora v řídícím algoritmu.

Možným rozšířením implementovaného simulačního modelu osobní železniční stanice je výměna libovolného komponentu. Celý simulační model byl již od začátku navržen pro snadnou výměnu komponentů.

Algoritmus řízení implementovaný ve výpočetním jádru je zobecněný, takže lze implementovat libovolný simulační model s dodržením základních implementačních vzorů. Výpočetní jádro obsahuje obecné vzory agenta a komponentu, které se dají využít pro vytváření dalších komponentů s konkrétní činností.

Závěr

Cílem teoretické části bylo seznámení s architekturou agentově orientovaných simulačních modelů a řídícím algoritmem pro řízení modulů diskrétní simulace, spojité simulace a animace.

V praktické části byla popsána implementace výpočetního jádra s podporou on-line animace a simulačního modelu osobní železniční stanice. Implementace obou částí splňuje kladené požadavky a jsou snadno rozšířitelné.

Výpočetní jádro, včetně simulačního modelu osobní železniční stanice, bylo implementováno jako knihovna a použito při implementaci simulačního nástroje [5] s podporou on-line animace.

Výpočetní jádro a simulační model byly otestovány na datech osobní železniční stanice Praha hlavní nádraží a splnily veškeré požadavky na ně kladené.

Simulační nástroj založený na implementovaném jádru se simulačním modelem osobní železniční stanice bude využit pro testování komponentu [13], který implementuje vyhledání nejvhodnější nástupištní koleje pro vlak, jehož plánovaná kolej je obsazena.

Obsah přiloženého datového média

Přiložené datové médium obsahuje:

- aplikace Simulační nástroj a Definice vlaků (aplikace pro definici seznamu vlaků),
- zdrojové kódy výpočetního jádra se simulačním modelem osobní železniční stanice a
- přílohy – popis výpočetního jádra a simulačního modelu osobní železniční stanice (ve formě nápovědy) a diagramy (tříd, komunikační).

Seznam použité literatury

1. KAVIČKA, Antonín; KLIMA, Valent; ADAMKO, Norbert. *Agentovo orientovaná simulácia dopravných uzlov*. Žilina : Žilinská univerzita, 2005. 206 s. ISBN 80-8070-477-5.

2. Jennings, N., R.: *An agent-based approach for building complex software systems*, Communications of the ACM, April 2001/Vol. 44, No. 4, pp. 35-41

3. Nwana, H., S.: *Sofware Agents: An Overview*, Knowledge Engineering Review 11 (3), 1996, pp. 2005-244

4. Kavička, Antonín: *Pokročilé techniky modelování a simulace: Diskrétní simulace*, Učební text Fakulty elektrotechniky a informatiky Univerzity Pardubice, Univerzita Pardubice, Pardubice 2009

5. JEŽEK, Ladislav. *Simulační nástroj železniční stanice založený na agentově orientovaném výpočetním jádru*. Pardubice, 2010. 55 s. Diplomová práce. Univerzita Pardubice, Fakulta elektrotechniky a informatiky.

6. *SŽDC : Správa železniční dopravní cesty* [online]. 2010-06-13 [cit. 2010-08-04]. Knižní jízdní řády. Dostupné z WWW: <http://www.szdc.cz/provozovani-drahy/knizni-jizdni-rady-130610.html>.

7. NAGEL, Christian, et al. *C# 2008 : Programujeme profesionálně*. Vydání první. Brno : Computer Press, 2009. 1126 s, 772 s. ISBN 978-80-251-2401-7.

8. MAREŠ, Amadeo. *1001 tipů a triků pro C#*. Vydání první. Brno : Computer Press, 2008. 360 s. ISBN 978-80-251-2125-2.

9. KAVIČKA, Antonín; JÁNOŠÍKOVÁ, Ľudmila. Modelovanie koľajiska a výpočet najkratšej jazdnej cesty. *Komunikácie : Vedecké listy Žilinskej univerzity*. 1999, 2, s. 9-21. ISSN 1335-4205.

10. Dijkstrův algoritmus. In *Wikipedia : the free encyclopedia* [online]. St. Petersburg (Florida) : Wikipedia Foundation, 25. 6. 2006, last modified on 1. 7. 2010 [cit. 2010-08-05]. Dostupné z WWW: <http://cs.wikipedia.org/wiki/Dijkstr%C5%AFv_algoritmus>.

11. KAVIČKA, Anotním; BAŽANT, Michael. Návrh infrastruktury železničních uzlů s podporou počítačové simulace : (část 1). *AUTOMA : časopis pro automatizační techniku*. 2007, 5, s. 2-3. Dostupný také z WWW: <http://www.odbornecasopisy.cz/index.php?id_document=34373>.

12. KAVIČKA, Anotním; BAŽANT, Michael. Návrh infrastruktury železničních uzlů s podporou počítačové simulace : (část 2). *AUTOMA : časopis pro automatizační techniku*. 2007, 6, s. 2-4. Dostupný také z WWW: <http://www.odbornecasopisy.cz/index.php?id_document=34407>.

13. BAŽANT, Michael, KAVIČKA, Antonín. Artificial neural network as a support of platform track assignment within simulation models reflecting passenger railway stations. *Journal of Rail and Rapid Transit*. 2009, vol. 223, no. 5, s. 505-515.

14. KAVIČKA, Antonín, KLIMA, Valent, ADAMKO, Norbert. *Agentovo orientovaná simulácia dopravných uzlov*. Žilina: EDIS - vydavateľstvo ŽU, 2005. 206 s. Monografie. ISBN 80-8070-477-5.

15. ADAMKO, Norbert, KAVIČKA, Antonín, KLIMA, Valent. *DAAAM International Scientific Book 2007*. Branko Katalinic. Vienna : DAAAM International Publishing, 2007. ISBN 3-901509-60-7. Agent based simulation of transportation logistic systems, s. 407-422.