UNIVERZITA PARDUBICE

Fakulta elektrotechniky a informatiky

Simulační jádro se zaměřením na dopravní systémy

Štěpán Moravčík

Bakalářská práce

2010

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

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Štěpán MORAVČÍK**
Osobní číslo:	**I07722**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Simulační jádro se zaměřením na dopravní systémy**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem BP je:
- vytvoření vzorového simulačního jádra založeného na metodě plánování událostí v programovacím jazyce Java,
- na vytvořeném simulačním jádře realizovat simulační model opravny železničních vozů s uvažováním zdrojů různých typů a náhodně vstupujících entit typu železniční vůz,
- návrh úloh pro předmět Modelování a simulace spočívajících v zadání dílčích úkolů pro studenty (např. realizace generátorů vstupních entit, doby trvání jednotlivých úkonů apod.).

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

***Kavička A. Modelování a simulace. Elektronické sylaby k předměu, Univerzita Pardubice, 2009.**
***Křivý I., Kindler E. Simulace a modelování. Elektronická skripta Ostravské univerzity, 2001, 146 s.**
***Kelton W. Simulation with Arena. New York: McGraw-Hill, 2004, 668 s. ISBN: 0-07-291981-7.**

Vedoucí bakalářské práce: **Ing. Michael Bažant, Ph.D.**
Katedra softwarových technologií

Datum zadání bakalářské práce: **15. ledna 2010**

Termín odevzdání bakalářské práce: **14. května 2010**

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

Ing. Lukáš Čegan, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2010

Prohlášení autora

Prohlašuji, že jsem tuto práci 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 13. 08. 2010 Štěpán Moravčík

Anotace

V této práci byla realizována vzorová implementace simulačního jádra založeného na metodě plánování diskrétních událostí. Práce by měla zároveň sloužit jako podpora při studiu předmětů zabývajících se problematikou modelování a simulace.

Klíčová slova

simulace, simulační jádro, diskrétní událost, prioritní fronta

Title

Simulation engine related to transport systems.

Annotation

In this work is realized the exemplary implementation of simulation engine based on discrete event-scheduling method. Work should be used as a study support for subjects, which are concerned with problematic of modeling and simulation.

Keywords

simulation, simulation engine, discrete event, priority queue

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Hlavním cílem této práce je vytvoření funkčního simulačního jádra založeném na metodě plánování diskrétních událostí.

V první části jsou vysvětleny základní pojmy z oblasti simulace a datových struktur. Jejich pochopení usnadňuje pochopení fungování celého realizovaného jádra. Z oblasti simulace je nejprve vysvětleno, co je samotná simulace, k čemu se využívá abstrakce a co je událost. Dále pak, co je považováno za simulační pokus a projekt, takto se dostaneme až k vysvětlení pojmu simulační jádro a k popsání metod používaných pro synchronizaci simulačních výpočtů v něm probíhajících. Větší pozornost je věnována metodě plánování diskrétních události, která se uplatňuje ve vytvořené aplikaci.

V další části je popsána vytvořená demonstrační aplikace nazvaná Simulátor opravny železničních vozů. Je napsána v programovacím jazyce Java v prostředí NetBeans verze 6.9. Vysvětleno je, jak fungování programu uvnitř (samotný běh jádra, výběr událostí z kalendáře a následné vykonávání), tak ovládání programu uživatelem. Pro názornost je s aplikací realizován simulační projekt zabývající se určením počtu obslužných zdrojů v opravně železničních vozů.

Poslední část se věnuje dalšímu bodu zadání práce, a to vytvoření úloh pro studenty předmětu modelování a simulace. Jedná se o dva úkoly, se kterými by si studenti měli bez problémů poradit. První spočívá v doplnění generátorů pseudonáhodných čísel s různými rozděleními pravděpodobností (normální, exponenciální, rovnoměrné) do simulátoru. Ve druhém je úkolem studentů určit optimální počty zdrojů a kapacitu odstavných kolejí v plánované opravně železničních vozů.

2 Základní pojmy

V této části jsou vysvětleny základní pojmy vyskytující se v práci. Základní pojmy týkající se simulace vycházejí z použité literatury [4, 5].

2.1 Simulace

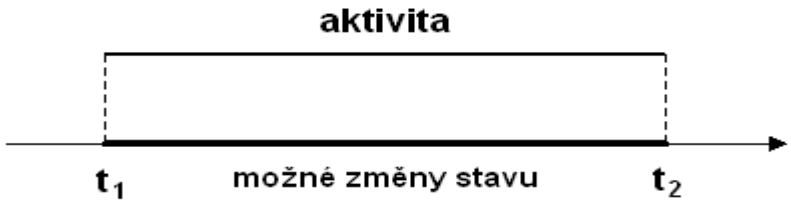
Simulace je výzkumná metoda, při které je určitý dynamický systém z reálného světa nahrazen simulačním modelem (simulátorem) za účelem provádění experimentů s tímto modelem a získání informací o původním systému. Získané informace mohou přispět k optimalizaci systémů, které již v realitě fungují nebo při výstavbě těch, které by mohly existovat v budoucnu. Dynamický systém je takový systém, ve kterém nezanedbáváme význam času. Nejrozšířenějším typem simulace je simulace využívající výpočetní techniku, čili simulace číslicová. Uplatňuje se při zkoumání složitějších systémů, které je nutné zkoumat s velkým počtem měnících se parametrů.

2.1.1 Abstrakce

Abstrakce se uplatňuje při tvorbě simulačního modelu podle předlohy ve skutečném světě. Při tvorbě je použita pouze část vlastností reálných objektů, která nejvíce ovlivňuje chování daných objektů v simulačním modelu. Například při simulování autobusové dopravy nás bude na jednotlivých vozech zajímat jejich kapacita nebo rychlost, ale údaj o jejich barvě nebo výrobci můžeme zanedbat, protože chování v daném modelu neovlivňuje vůbec nebo pouze minimálně.

2.1.2 Aktivity

Aktivita je základní akční jednotkou simulace. Představuje jistou činnost v simulačním modelu. Každá aktivita má čas svého trvání a mění stav v systému. Běh simulačního modelu pak znamená množství po sobě jdoucích aktivit ve stejném pořadí jako ve skutečném systému. Aktivity dělíme z hlediska času, ve kterém vykonávají změny stavu systému, na spojité a diskrétní. Spojité aktivity jsou takové, které mohou měnit stav po celou dobu jejich výskytu. Tedy od času t_1, ve kterém aktivita začíná, až po čas t_2, ve kterém končí.



Obr. 1 Spojitá aktivita

Diskrétní aktivity mohou stav měnit pouze po jejich dokončení, tedy v čase t_2. Ukončení diskrétní aktivity se změnou stavu systému nazýváme událost (U).

Obr. 2 Diskrétní aktivita

Posloupnost za sebou jdoucích aktivit, které spolu tvoří logický celek, nazýváme proces.

2.1.3 Simulační pokus

Simulační program je program, který řídí výpočty a běh simulace. Tento program je spouštěn v rámci simulačního pokusu (experimentu) s různými konfiguracemi, kterým se říká scénáře. Scénář chápeme jako popis dynamického chování simulačního modelu, jenž obsahuje:

- scénu, kterou chápeme jako soubor stálých prvků systému s hodnotami jejich parametrů,
- pravidla pro generování, vstup a výstup dočasných prvků a
- rozhodující a řídící algoritmy popisující aktivity simulátoru.

Scénář tvoří množinu vstupních dat, ze kterých pomocí simulačního programu dokážeme získat data výstupní, charakterizující chování systému. Jeden běh simulačního programu s takto vymezeným scénářem, při kterém zkoumáme chování vymezeného systému, nazýváme simulační pokus. Jelikož jsou vstupní parametry často představovány náhodnými proměnnými, jsou i výstupní parametry realizace náhodných proměnných. V těchto případech je nutné provést několik běhů simulačního programu se stejným scénářem, ale jinými hodnotami vstupních proměnných generovaných podle stejných pravidel. Tyto běhy pak nazýváme replikace pokusu, z nichž získáváme výstupní hodnoty.

Výstupní data po dostatečném počtu replikací statisticky zpracujeme a získáme obraz o chování systému s daným scénářem.

2.1.4 Simulační projekt

Simulační projekt lze chápat jako sérii simulačních pokusů s měnícím se scénářem. Cílem je nalezení takového scénáře, který je podle zadaných kritérií nejvíce vyhovující. Je to tedy optimalizační metoda, která nenabízí automaticky optimální řešení, ale poskytuje odpovědi, podle nichž lze vhodnou změnou scénáře optimální řešení nalézt.

2.1.5 Synchronizace simulačního výpočtu

Při běhu simulačního programu se mění čas, který je odrazem času ve vymezeném systému v reálném světě. Tento čas nazýváme simulační a je potřeba zajistit, aby aktivity vykonávané v simulátoru závisely na simulačním čase stejně, jako aktivity vykonávané v reálném světě závisí na toku reálného času. Což znamená i to, že čas plyne vždy kupředu, nikdy ne zpět. V simulátoru většinou simulační čas plyne rychleji než skutečný, aby bylo možné pozorovat jeho chování v delším časovém období. Úsek výpočtu, při kterém se nemění simulační čas, nazýváme simulační krok. Synchronizaci simulačního výpočtu lze zajistit řadou algoritmů, níže jsou uvedeny základní čtyři přístupy.

a) Metoda plánování událostí

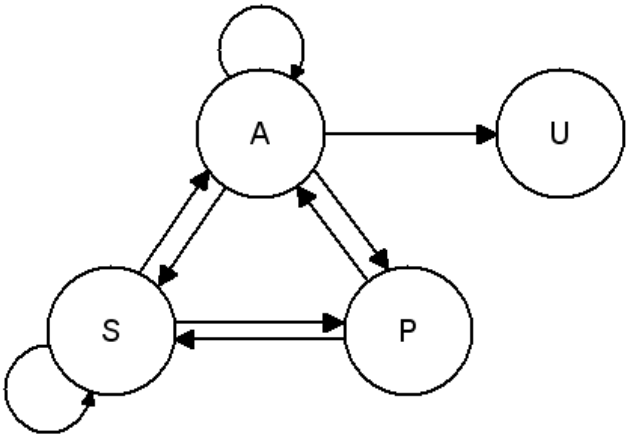
je nejrozšířenější metodou synchronizace diskrétní simulace. Je založena na plánování událostí do budoucnosti. Jelikož je tato metoda použita v demonstrační aplikaci, bude podrobněji vysvětlena později.

b) Metoda interakce procesů

je metoda využívaná u diskrétní simulace. Částečně je založena na metodě plánování událostí. Každý proces je zde rozdělen na jednotlivé aktivity s ním spojené. Pokud je proces aktivován, provede se pouze aktuální aktivita procesu spojená se změnou stavu systému. Proces se může nacházet v jednom z následujících stavů:

- aktivní – proces je právě obsluhován, je prováděn výpočet
- ukončený – proces je ukončený, již nebude obsluhován
- suspendovaný – vykonávání procesu je naplánováno, proces čeká
- pasivní – proces čeká, jeho vykonávání není naplánováno.

Při návrhu simulačního modelu založeném na této metodě se pracuje přímo s mechanismem změn stavů procesů.



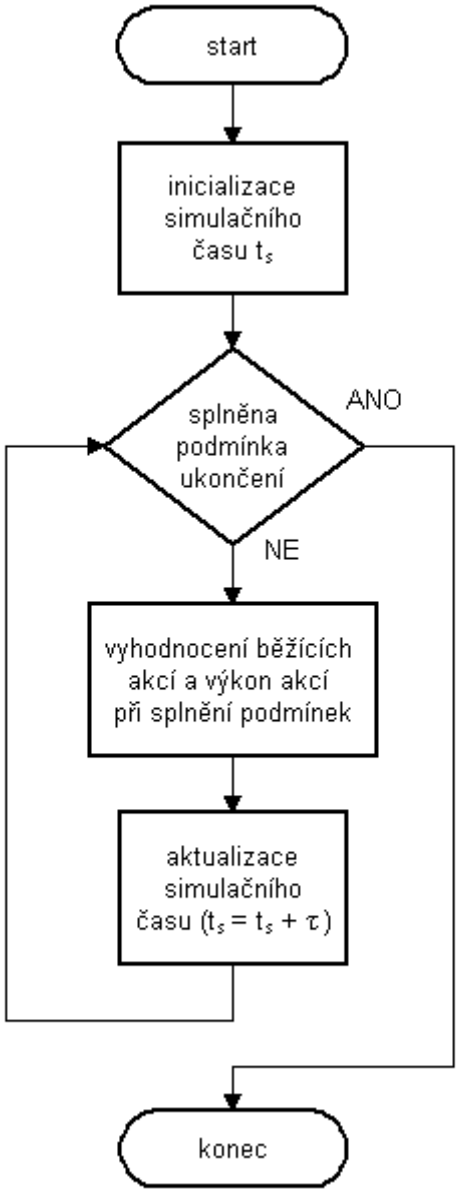
Obr. 3 Stavy procesu

c) Metoda snímání aktivit

také označována jako metoda dvoufázová. Lze jí použít jak při realizaci diskrétní simulace, tak i pro simulaci spojitou. Principem je rozdělení simulačního běhu na části po časových úsecích τ, které jsou obvykle stejně dlouhé, tzv. ekvidistantní. V každém takovém úseku dochází k vyhodnocování právě běžících aktivit, zda v simulačním čase t_s odpovídajícím hranici úseku nedošlo:

- k výskytu koncové události u diskrétní simulace nebo
- ke splnění aktivační podmínky k vykonání určité akce u spojité simulace.

Přesnost této metody závisí na vhodném zvolení velikosti parametru τ. Jelikož jsou časové úseky mezi jednotlivými časy vyhodnocování stejně dlouhé, mluvíme o synchronní metodě.

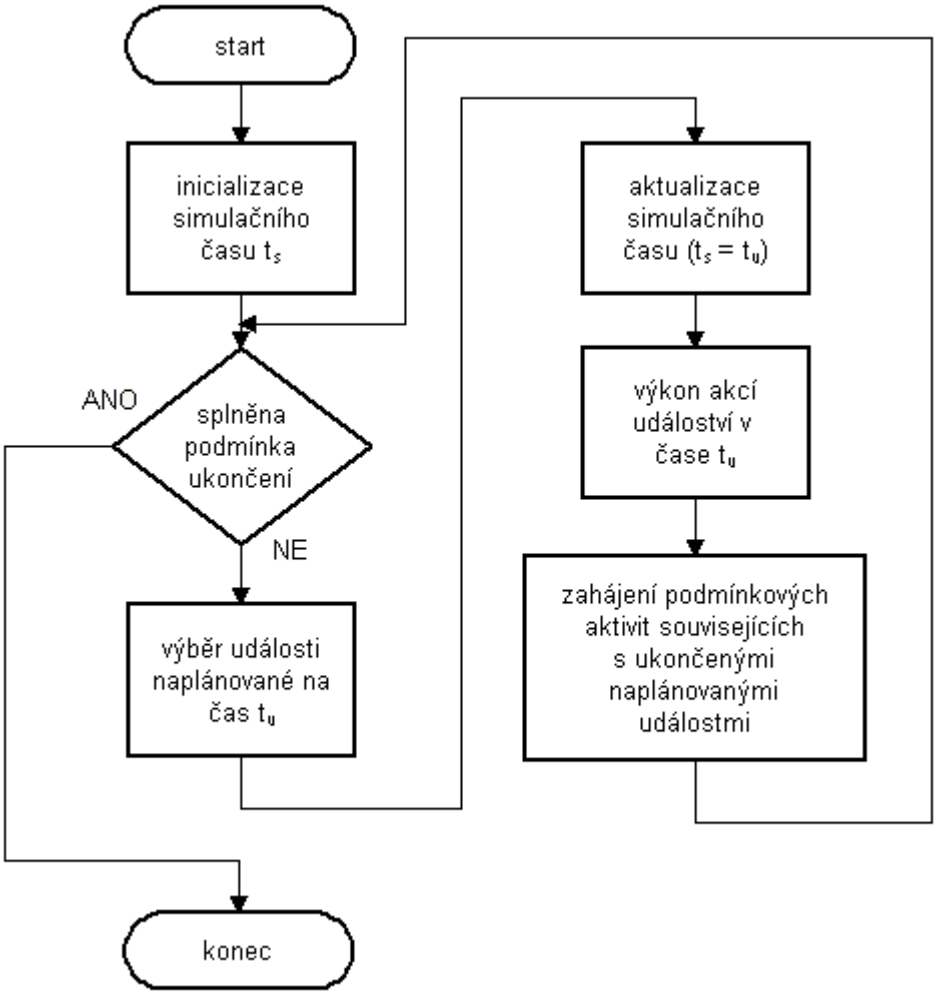


Obr. 4 Metoda snímání aktivit

d) Třífázová metoda

je metoda používaná pro diskrétní simulaci. Rozlišuje dva druhy aktivit. Aktivity plánované, které se plánují předem a čas jejich vykonávání je předem znám, a aktivity podmínkové, jejichž výskyt je dán určitou aktivační podmínkou. V každém simulačním kroku se nejprve obslouží aktuální plánovaná událost, tedy událost s nejnižším časem výskytu t_u, a poté je testováno, zda nebyla splněna podmínka pro některou z čekajících podmínkových aktivit. Pokud ano, tak se podmínková aktivita vykoná. Nejčastějším

případem je, že podmínkové aktivity čekají na uvolnění zdrojů aktivitami plánovanými, tím se lze vyhnout soupeření dvou aktivit o stejný druh zdroje.



Obr. 5 Třífázová metoda

2.1.6 Simulační jádro

Modul, řídící běh celého simulačního programu, ve kterém jsou implementovány algoritmy založené na výše uvedených metodách, nazýváme simulační jádro. Simulační jádro také obsahuje operace umožňující zasahovat do nastavení proměnných, podle kterých se mění dynamika modelu.

2.2 Datové struktury

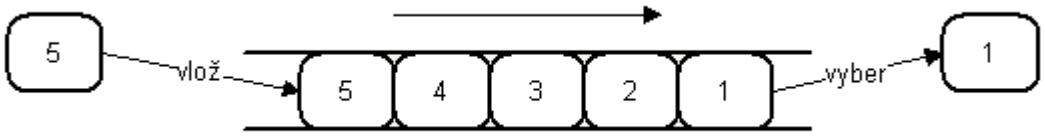
Datové struktury představují způsob uchovávání informací. Zjednodušují práci s informacemi, zajišťují přístup k nim a možnost jejich modifikace. Správně zvolené datové struktury snižují nejen náročnost výpočtů v programu, ale i velikost obsazeného paměťového prostoru. Existuje velké množství typů datových struktur, každý z nich má své klady a zápory, a proto je potřeba při návrhu aplikace zvolit správný typ s nejvíce vyhovujícími vlastnostmi. Ve vyšších programovacích jazycích jsou tyto struktury většinou již vytvořené a připravené k použití. Základní pojmy v části věnující se datovým strukturám vycházejí z použité literatury [3].

V souvislosti s problematikou datových struktur se používají pojmy abstraktní datový typ (ADT) a abstraktní datová struktura (ADS). Abstraktní datový typ je matematická struktura, která se skládá z alespoň jedné třídy matematických objektů (domény) a z alespoň jedné operace pracují s prvky těchto tříd (domén). Abstraktní datová struktura je pak konkrétní realizací abstraktního datového typu.

2.2.1 Fronta

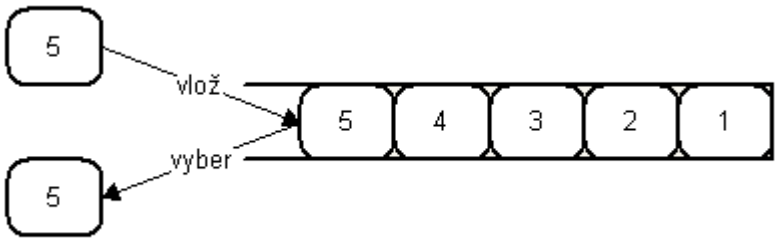
Je jeden ze základních typů datových struktur. Vkládání a vybírání prvků ze struktury umožňuje jedna vstupně/výstupní brána. Nejčastěji se lze setkat s frontou realizovanou s využitím dynamického lineárního spojového seznamu nebo statického pole. Můžeme se sekat s frontami dvou typů.

- **Fronta typu FIFO** (first in, first out) – v překladu první dovnitř, první ven. Vybírání prvků probíhá ve stejném pořadí jako jejich vkládání, prvek, který byl vožen první, bude také první odebrán. Jedná se tedy o frontu, kterou známe z reálného světa například z obchodů.



Obr. 6 Fronta FIFO

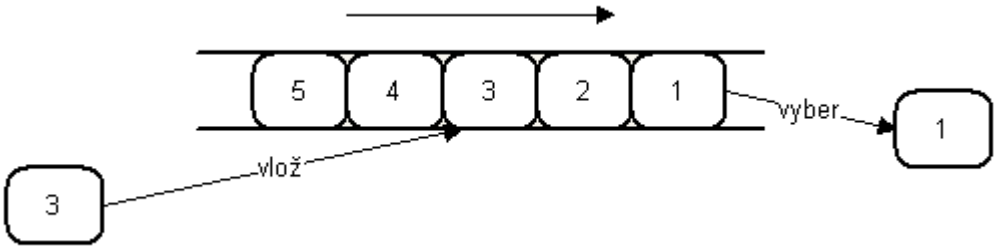
- **Fronta typu LIFO** (last in, first out) – Nazývaná také jako zásobník. V překladu poslední dovnitř, první ven. Jde tedy o zásobník, kde prvním vybíraným prvkem je ten, který byl vložen jako poslední. Kdybychom chtěli vybrat prvek vložený jako první, museli bychom nejdříve vybrat všechny ostatní prvky, které byly vloženy po něm.



Obr. 7 Fronta LIFO

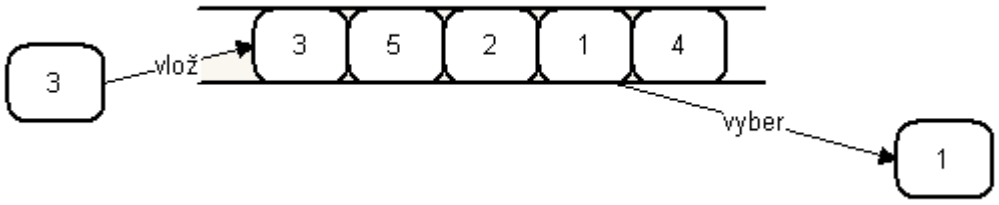
2.2.2 Prioritní fronta

U prioritní fronty nezávisí pořadí vybíraných prvků na pořadí, ve kterém do fronty přicházely. Pořadí vybírání prvků určuje priorita, která je nastavena každému prvku. Při vybírání z fronty je vždy vybrán prvek s nejvyšší (nejnižší) prioritou. Z hlediska implementace jsou možné dva způsoby přístupu, buď jsou prvky ve struktuře uspořádány nebo jsou vkládány podle jejich příchodu, čili neuspořádaně. V prvním případě je při každém vkládání nutno projít strukturu a zjistit, na které místo přicházející prvek zařadit. Při výběru pak pouze odebereme prvek s nejvyšší prioritou, tedy prvek na prvním místě.



Obr. 8 Prioritní fronta utříděná

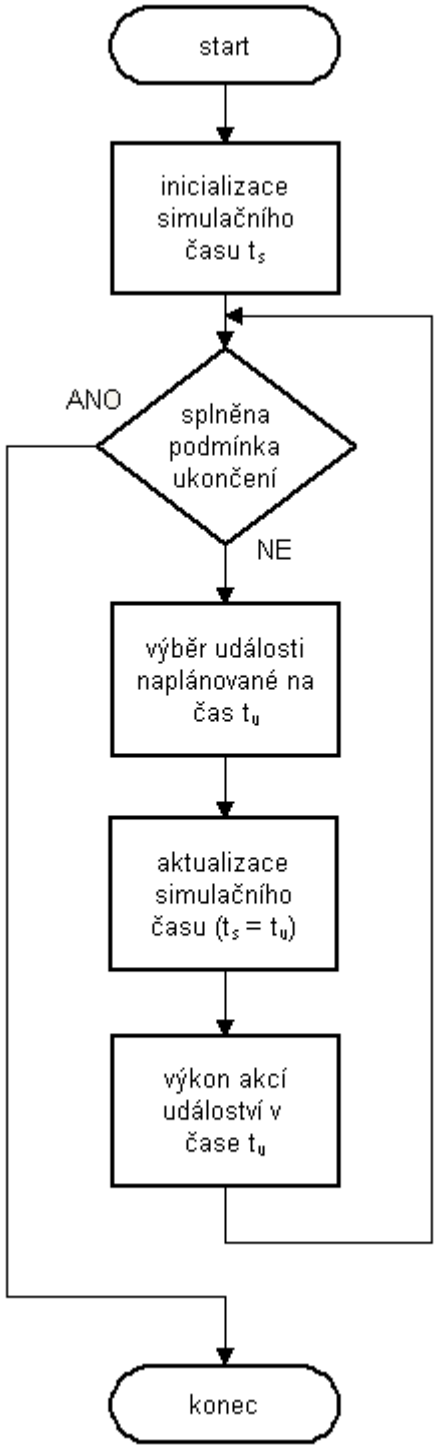
V případě druhém, kdy je struktura neuspořádaná, probíhá vkládání ihned, bez vyhledávání. Při výběru je pak nutné projít celou strukturu a vyhledat prvek s nejvyšší prioritou a ten poté odebrat.



Obr. 9 Prioritní fronta neutříděná

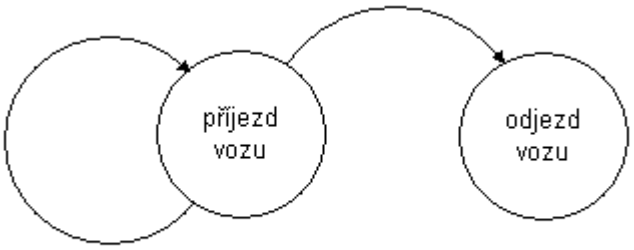
2.3 Metoda plánování diskrétních událostí

Tato metoda synchronizace simulačního výpočtu je jedna z nejčastěji používaných. Využívá diskrétních událostí uložených v kalendáři událostí realizovaném prioritní frontou. V kalendáři jsou uložené naplánované události až do doby jejich výskytu. Simulační výpočet běží v cyklu, ve kterém je vybrána vždy událost s nejvyšší prioritou, tedy s nejnižší hodnotou času výskytu, simulační čas se nastaví podle času jejího výskytu a vykonají se akce spojené s jejím výskytem [4].



Obr. 10 Diagram diskrétní simulace

Události do kalendáře jsou přidávány průběžně, protože některé události při svém vykonávání zároveň plánují výskyt následných událostí. Graficky znázornit vazby mezi diskrétními událostmi umožňuje diagram plánování událostí.



Obr. 11 Diagram plánování událostí

Na příkladu je ukázána situace příjezdu a odjezdu vozu. Při příjezdu vozu je naplánován nejen jeho odjezd, ale i příjezd vozu dalšího. Při odjezdu se další události neplánují.

3 Demonstrační aplikace

3.1 Popis

Demonstrační aplikace se zabývá simulací opravny železničních vozů, založené na metodě plánování diskrétních událostí. Opravna má několik stanovišť, na nichž oprava vozů probíhá. Na každém z nich je využíván jeden ze dvou zdrojů. Uživatel má možnost nastavit parametry modelu, kterými jsou časy potřebné k dokončení oprav na jednotlivých stanovištích včetně jejich rozdělení pravděpodobnosti, počet zdrojů, délka fronty před halou a kapacita haly. Při běhu simulace je možno sledovat jednotlivé kroky a průběžné výsledky. Aplikace je vytvořena v jazyce Java ve vývojovém prostředí NetBeans verze 6.9. Diagram tříd je přiložen v příloze [A].

3.1.1 Simulační jádro

Hlavní částí celé aplikace je třída simulačního jádra nazvaná SimJadro, která zajišťuje běh simulačního výpočtu. Třída vypadá následovně:

```
public class SimJadro implements ISimJadro {

    private IStavSimJadra stav = new StavSimJadra();
    private IUdalost aktualniUdalost;

    public void pripravNaBeh(Date zacatek) {...}
    public void beh(Date zacatek, Date konec) {...}
    public void dalsiKrok(Date konec) {...}
    public IStavSimJadra getStav() {...}
}
```

Atribut *stav* je instance třídy *StavSimJadra*, v této třídě se uchovávají informace o stavu simulačního jádra, které se při běhu simulačního výpočtu mění. Obsah třídy *StavSimJadra* bude vysvětlen později. Atribut *aktualniUdalost* představuje událost, která je právě zpracovávána. Při každém kroku se tato událost vybírá z kalendáře událostí, aby mohlo dojít k jejímu zpracování. Metoda *pripravNaBeh* připraví jádro pro běh od zadaného času. To znamená, že vynuluje všechna počítadla, nastaví simulační čas na začátek simulace, zajistí připravení prázdného kalendáře událostí a vloží do něj první událost. Metoda *beh* provede jednu replikaci simulačního běhu v zadaném časovém rozmezí. Nejdříve volá metodu *pripravNaBeh* a poté pracuje v cyklu:

```
while (!stav.prazdnyKalendar()) {
            aktualniUdalost = (IUdalost) stav.getUdalost();
            if (konec.getTime() != 0 &&
                aktualniUdalost.getCasVyskytu().after(konec)) {
                  break;
            }
            aktualniUdalost.vykonej(stav);
            stav.pridejDoZpravy(aktualniUdalost.vypis(stav));
        }
```

V každém kroku tedy vybírá událost, vykonává ji a zapisuje do zprávy řádek o jejím vykonání. Cyklus trvá, dokud není vyprázdněn kalendář událostí nebo dokud simulační čas nepřekročí čas konce simulace, je-li čas konce stanoven.

Metoda *dalsiKrok()* se metodě *běh* velice podobá, avšak vykoná vždy pouze jeden krok simulačního výpočtu. Používá se při běhu, při kterém se informace o činnosti jádra průběžně vypisují, aby je uživatel mohl sledovat. V cyklu se tedy volá tato metoda a poté se vypisují proměnné a statistiky odrážející stav simulačního běhu.

Poslední metoda jádra *getStav()* pak vrací instanci třídy *StavSimJadra*. Tato třída uchovává množství informací a funguje jako datová struktura, se kterou jádro pracuje. Pomocí ní zjišťuje a mění data, na kterých závisí běh simulačního výpočtu. Obsahuje následující atributy:

```
private PriorityQueue<IUdalost> kalendarUdalosti;
private Date simulacniCas;
private Queue<IVuz> frontaPredHalou;
private int pocetVozuVHale;
private int pocetPrijezdu;
private int pocetOdjezdu;
private int pocetNespokojenychOdjezdu;
private int maxDelkaFrontyPredHalou;
private int maxPocetVozuVHale;
private int maxPrijezdu;
private int prijezdVozuCas;
private int prijezdVozuOdchylka;
private int prijezdVozuRozdeleni;
private StringBuilder zprava;
private String posledniZprava;
private IStanoviste stanovisteStart;
private IStanoviste stanovisteMontaz;
private IStanoviste stanovisteSedadla;
private IStanoviste stanovistePodvozky;
private IStanoviste stanovisteDemontaz;
private Random generator;
```

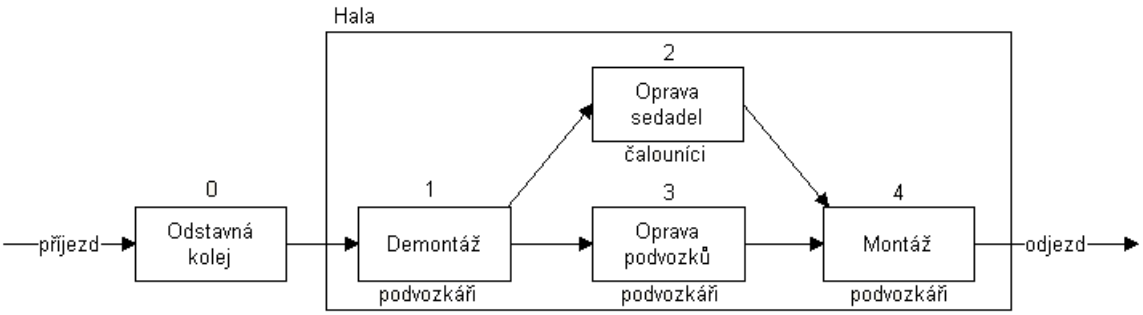
U většiny atributů je již z názvu patrné k čemu slouží a jak pracují. Parametry *maxDelkaFrontyPredHalou*, *maxPocetVozuVHale* a *maxPrijezdu* se při běhu simulačního výpočtu nemění, nastaví se na jeho začátku a určují vlastnosti modelu respektive délku simulace. *prijezdVozuCas, prijezdVozuOdchylka, prjizedVozuRozdeleni* představují

parametry, na jejichž základě probíhá generování času příjezdu dalšího vozu do opravny. Nejčastěji jde o exponenciální rozdělení pravděpodobnosti, avšak je možno nastavit také rozdělení normální či rovnoměrné. K tomuto generování se využívá *generator*, který poskytuje pseudonáhodná čísla. Parametry *zprava* a *posledniZprava* slouží k výpisu průběhu simulačního běhu. V každém kroku je generován jeden řádek zprávy, který obsahuje čas, číslo vozu, číslo stanoviště a informaci, zda se jedná o příjezd, či odjezd. Tento řádek je uchován v atributu *posledniZprava* a kompletní zpráva, do které se řádek přidává, je uložena v proměnné *zprava*. Z ní se na konci simulace vypíše.

Mimo popsané atributy obsahuje třída také množství metod pro práci s nimi. Například *addUdalost(IUdalost udalost)* slouží k naplánování další události do kalendáře událostí a *getUdalost()* naopak odebere z kalendáře událost s nejnižším časem výskytu. Popisovat zde všechny metody je zbytečné, protože z jejich názvů je jasné k čemu slouží. Metody, které stojí za zmínku, jsou metody pro generování náhodných čísel *dalsiNorm(double stred, double odchlka)*, *dalsiExp(double stred)* a *dalsiRovn(double min, double max)*. Tyto metody používají atribut *generator* a generují čísla určitého rozdělení. Normálního, exponenciálního nebo rovnoměrného, podle zadaných parametrů. Používají se při plánování výskytu další události.

3.1.2 Stanoviště opravny

Oprava vozů probíhá na několika stanovištích, pro přehlednost je to znázorněno na následujícím obrázku.



Obr. 12 Jednoduché schéma opravny

Celý proces opravy začíná příjezdem vozu na odstavnou kolej. Jestliže není v hale plno, tedy není dosaženo její maximální kapacity, vjíždí vůz do haly na stanoviště číslo jedna. Pokud je v hale již plno, zařadí se vůz do fronty a čeká na uvolnění místa v hale. Pokud je však již dosaženo maximální kapacity odstavné koleje, musí vůz opravnu opustit

neuspokojen, protože již není prostor, kam vůz umístit. Po vjetí na stanoviště označené demontáž je vůz rozdělen na dvě části, a to sedadla a podvozek. (V aplikaci je toto rozdělení provedeno pouze tím, že jeden celý vůz putuje na stanoviště číslo dva a druhý na stanoviště číslo tři.) Tyto části pokračují na další stanoviště samostatně a po provedení jejich oprav na stanovištích číslo dvě a tři jsou ve stanovišti montáže opět spojeny a opouští halu, tedy i opravnu vcelku.

Jak je vidět na obrázku, opravu celého vozu zajišťují dva typy dělníků, a to podvozkáři na stanovištích montáže, demontáže a opravy podvozků a čalouníci na stanovišti opravy sedadel. Počty dělníků může uživatel aplikace upravovat.

Každé stanoviště je v aplikaci reprezentováno třídou *Stanoviste*, která obsahuje následující atributy:

```
private int cislo;
private int casPraceStred;
private int casPraceOdchylka;
private int casPraceRozdeleni;
private int pocetPracovniku;
private long odpracovanyCas;
private Date pracujeSeDo;
```

Atribut *cislo* udává číslo stanoviště, podle kterého se plánují další události. Je také použito při výpisu do zprávy. Trojice atributů *casPraceStred*, *casPraceOdchylka* a *casPraceRozdeleni* jsou parametry, dle kterých se generuje čas práce pracovníků, čili čas strávený vozem ve daném stanovišti. *pocetPracovninku* udává počet pracovníků na daném stanovišti. Rostoucí počet pracovníků znamená klesající dobu potřebnou pro vykonání opravy. Znamená to, že dva pracovníci potřebují na zadanou práci poloviční dobu, než by potřeboval jeden pracovník. Pro potřeby statistiky je zde zaveden i atribut *odpracovanyCas*, s jehož pomocí lze vypočítat procentuální vytížení pracovníků na daném stanovišti. Poslední atribut *pracujeSeDo* uchovává informaci o čase, ve kterém skončila nebo skončí poslední oprava prováděná na daném stanovišti. Díky tomu se nemůže stát, že by na jednom stanovišti byly přítomny dva vozy naráz. Mimo těchto atributů třída obsahuje metody určené pro jejich obsluhu.

Stanoviště číslo čtyři, nazvané montáž, se od ostatních stanovišť liší tím, že montuje dohromady dvě části vozu (v aplikaci reprezentované dvěma vozy se shodným číslem). Proto navíc obsahuje seznam vozů čekajících na kompletaci, ze kterého je buď příslušný vůz vybrán a jako celek opouští simulaci, nebo je do seznamu vůz zařazen a čeká na jeho

druhou část. Toto stanoviště je reprezentováno třídou *StanovisteCekaci*, které je potomkem třídy *Stanoviste*, pouze navíc obsahuje proměnnou *seznamCekajicich* typu *ArrayList<IVuz>* a metody s ní pracující *vlozNaSeznamuCekajicich(IVuz vuz)* a *vyberZeSeznamuCekajicich(IVuz vuz)*. První metoda pouze vloží vůz do seznamu, druhá metoda odebere vůz ze seznamu v případě, že se v něm již nachází, nebo ho do seznamu zařadí.

3.1.3 Události

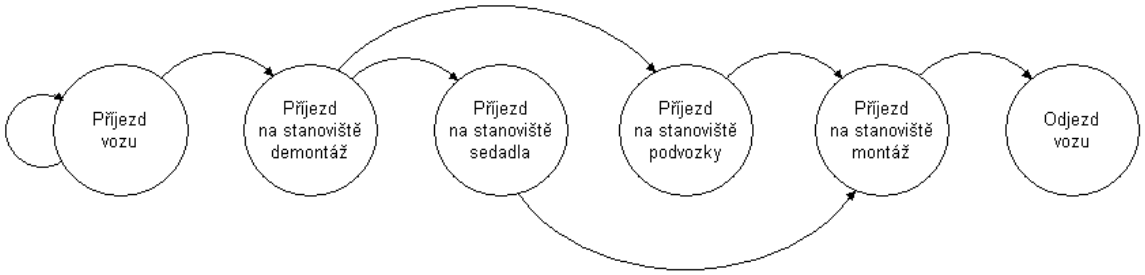
Běh celého simulačního výpočtu je posloupnost událostí. Události se uchovávají v kalendáři událostí, odtud se také vybírají a následně provádějí. V programu jsou všechny události reprezentovány potomky abstraktní třídy *Udalost*. Ta má dva potomky, a to třídy *Prijezd* a *Odjezd*. Třída *Udalost* vypadá následovně:

```
public abstract class Udalost implements IUdalost {

    protected Date casVyskytu;
    protected IVuz vuz;
    protected IStanoviste stanoviste;

    public Udalost(Date casVyskytu,IVuz vuz,IStanoviste stanoviste) {...}
    public String vypis() {...}
    public int compareTo(Object o) {...}
    public void vykonej(IStavSimJadra stav) {...}
    public Date getCasVyskytu() {...}
}
```

Význam atributů je z jejich názvů patrný, čas výskytu události, vůz a stanoviště spojené s událostí. Třída *Vuz* obsahuje pouze číslo vozu. Parametrický konstruktor také není nutné více popisovat. Metoda *vypis()* poskytuje řetězec znaků, který se přidává do zprávy a poté do výpisu pro uživatele. Metoda *compareTo()* je zde z důvodu ukládání událostí do prioritní fronty. Při vkládání je nutné rozhodnout na jaké místo se má vkládaná událost zařadit. Proto se událost postupně porovnává s událostmi ve frontě, dokud se nenajde příslušné místo. Porovnávání se provádí právě pomocí metody *compareTo(),* která porovnává události podle času jejich výskytu. Poslední metoda vrací čas výskytu dané události. Je použita při porovnávání při řazení do fronty.

Nejdůležitější metodou u událostí je metoda *vykonej()*, která zajišťuje synchronizaci simulačního času (nastavení simulačního času na čas výskytu události) a vykonání všech akcí spojených s danou událostí. U příjezdu nového vozu je to například naplánování příjezdu tohoto vozu na další stanoviště, ale také naplánování příjezdu dalšího vozu. Tyto

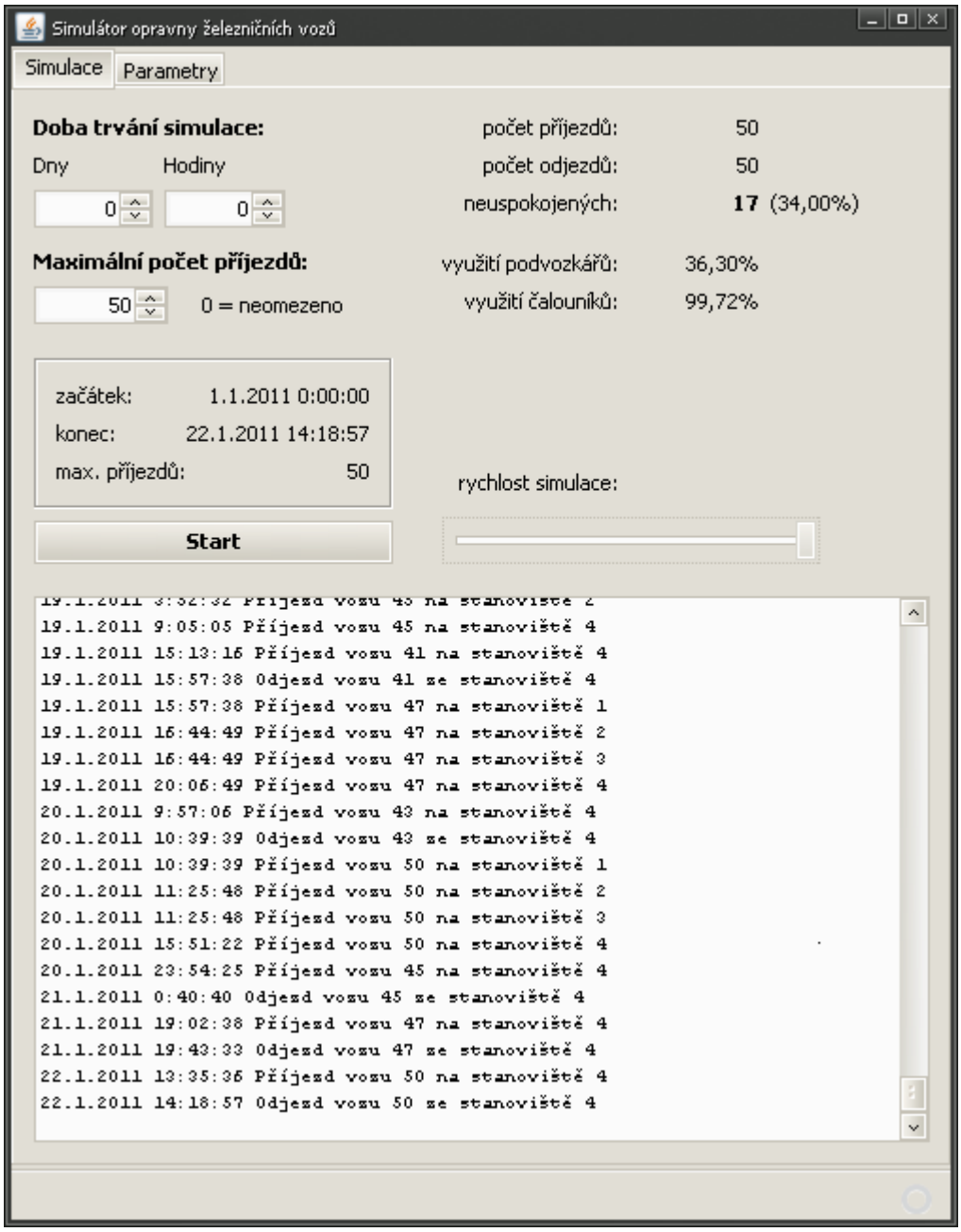
akce zajišťují metody *naplanujNasledujiciUdalost()* a *naplanujNovyVuz()* ve třídě *Prijezd.* V první z nich, se rozhoduje na základě parametrů jádra a stanoviště, na kterém se událost vyskytla, jaký další příjezd se naplánuje, na jaké stanoviště a na jaký simulační čas. Druhá metoda pak zajišťuje naplánování příjezdu dalšího vozu v simulačním čase určeném podle parametrů jádra. Obě tyto metody jsou volány z metody *vykonej()* třídy *Prijezd.* Třída *Odjezd* již žádné další události neplánuje, pouze přidává řádek do zprávy výpisu. *Diagram* plánování události graficky znázorňuje, jak na sebe události v modelu navazují.



Obr. 13 Diagram plánování událostí opravny

3.2 Ovládání

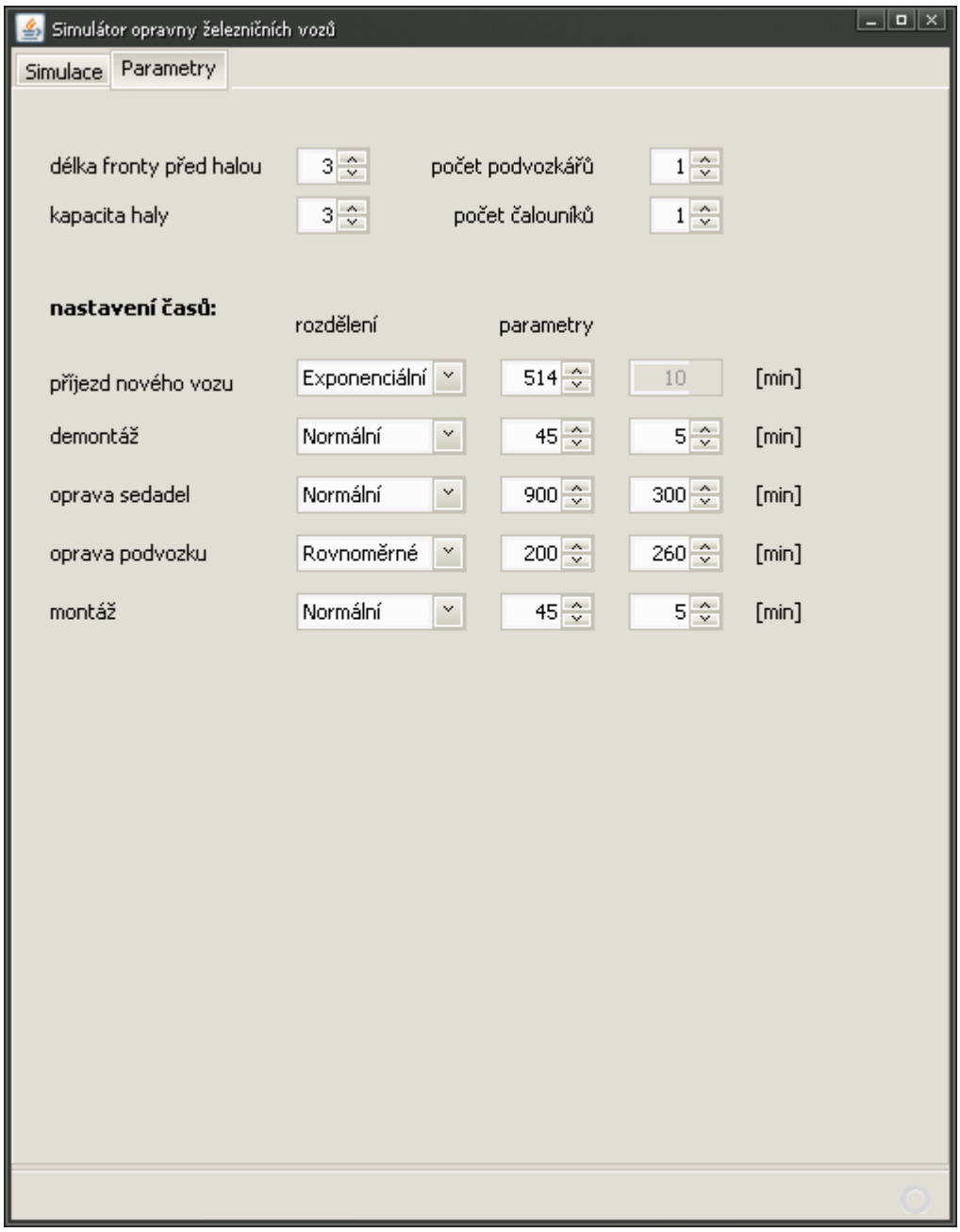
Aplikace je vybavena grafickým uživatelským rozhraním, proto práce s ní není nijak složitá. Po spuštění aplikace se zobrazí hlavní okno s možností spuštění simulace (viz. Obr. 12). V tomto okně se nastavuje dobra trvání simulace a to dvěma způsoby. Buď je doba uvedena časově, tedy jsou vyplněna pole *hodiny* a *minuty* a simulace běží až do zadaného času, nebo je vyplněno pole *maximální počet příjezdů* a simulace skončí v čase, kdy odjede poslední vůz. Jsou-li vyplněna všechna pole, tedy *doba trvání simulace* i *maximální počet příjezdů*, pak simulace končí při výskytu první z těchto podmínek. Po nastavení doby simulace je možné simulaci spustit tlačítkem *Start.* Po stisknutí tlačítka začíná simulační běh, probíhá simulační výpočet a jednotlivé události jsou vypisovány do pole umístěného ve spodní části okna, zároveň se v pravé části vypisují informace o počtech příjezdů a odjezdů, počtu a procentuálního poměru neuspokojených vozů a také procentuální využití dělníků. Výpis událostí zobrazuje datum, čas, jedná-li se o příjezd či odjezd, číslo vozu a číslo stanoviště. Začátek simulace je vždy nastaven na 1. ledna roku 2011. Rychlost simulace lze ovlivnit posuvníkem v pravé části od hodnoty jednoho kroku za jednu sekundu až po maximální rychlost, kterou aplikace dovoluje. Celé okno s již spuštěnou a ukončenou simulací je zobrazeno na obrázku 12.



Obr. 14 Hlavní okno aplikace

Nastavení doby trvání simulace a maximálního počtu příjezdů samozřejmě nemůže stačit pro provádění experimentů se simulátorem. Možnost nastavit další parametry modelu nalezneme pod záložkou *Parametry*, jak ukazuje obrázek 13. V horní části okna lze nastavit maximální délku fronty před halou, maximální kapacitu haly a počty jednotlivých dělníků. Níže je možné nastavit čas mezi příjezdy nových vozů a časy oprav na jednotlivých stanovištích. Hodnoty zde nastavené se používají ke generování časů, proto se zde nezadává pouze jedna hodnota, ale vybere se rozdělení pravděpodobnosti a vyplní se příslušné parametry. Na výběr jsou tři rozdělení pravděpodobností, a to rovnoměrné,

normální a exponenciální. Všechny hodnoty se udávají v minutách. Parametry, zadávané na základě výběru rozdělení pravděpodobnosti, jsou uvedeny v tabulce 1.



Obr. 15 Nastavení parametrů aplikace

Tab. 1 Parametry nastavení časů

Rozdělení pravděpodobnosti	První parametr	Druhý parametr
Rovnoměrné	minimum	maximum
Normální	střední hodnota	odchylka
Exponenciální	střední hodnota	-

3.3 Příklad

Pro názornou ukázku práce s vytvořeným simulátorem následuje řešení zadaného příkladu. Zadání vychází z použité literatury [2].

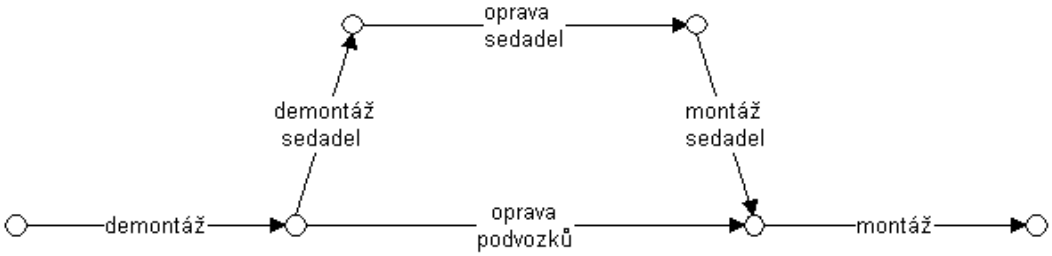
3.3.1 Zadání

Téma

Určení technického vybavení opravny železničních vozů.

Popis situace

V rámci restrukturalizace železničního podniku vyvstala potřeba vybudovat v pobočce Depa kolejových vozidel malou opravnu osobních železničních vozů. Předpokládejme, že prostoru není nazbyt, tzn. na kolej před opravnou se vejdou maximálně 3 vozy. V pořadí jak byly přistaveny, se tyto odebírají k opravě, přičemž se provádí běžné opravy podvozků a poničených sedadel. Opravy podvozků probíhají na speciální stolici, opravy sedadel v čalounické dílně – celá proces opravy je možné vyjádřit síťovým grafem na obrázku 16.



Obr. 16 Technologie opravy osobního vozu

V tabulce 2 jsou uvedeny jednotkové doby jednotlivých činností, u kterých se předpokládá vždy normální rozdělení pravděpodobnosti, a zdroje obsluhy. Tedy v případě zapojení více stejných zdrojů obsluhy se tyto doby dělí odpovídajícím počtem. Předpokládá se třísměnný provoz (tj. směny po 8mi hodinách bez přestávek).

Tab. 2 Jednotkové doby obsluhy jednotlivých činností

Název	μ	σ	Zdroje
(de)montáž podvozků	45 min.	5 min.	zvedák, dělník-podvozkář
oprava podvozku	4 hod.	1 hod.	dělník-podvozkář
(de)montáž sedadel	3 hod.	1 hod.	dělník-čalouník
oprava sedadel	16 hod.	5 hod.	dělník-čalouník

Vstupní proud je Poissonovský se střední hodnotou 2,8 vozu za kalendářní den.

Úkoly

Navrhněte celkovou koncepci dílny, resp. Určete počty jednotlivých obslužných zdrojů tak, aby nedocházelo k situaci, že (v průměru) nebude možné přistavit vůz k opravě. Samozřejmě, za předpokladu minimálních investičních i provozních nákladů.

3.3.2 Řešení

Než začneme nastavovat parametry simulátoru, je nutné zadanou úlohu mírně přizpůsobit, aby simulátoru vyhovovala a bylo možné ji na něm realizovat. To znamená především snížit počet stanovišť tak, aby odpovídal počtu stanovišť simulátoru. Tedy spojit tři stanoviště obsluhující sedadla ve stanoviště jedno, což je možně, neboť všechna tři stanoviště následují bezprostředně za sebou a využívají stejný zdroj obsluhy, tedy dělníky-čalouníky. Po spojení těchto stanovišť je také nutné určit dobu obsluhy na tomto stanovišti. Toho docílíme tak, že střední hodnoty třech původních stanovišť sečteme a získáme střední hodnotu upraveného stanoviště. Směrodatnou odchylku na výsledném stanovišti lze vypočítat jako odmocninu součtu druhých mocnin odchylek na původních stanovištích. Tímto se počet stanovišť zmenší na požadované čtyři.

Nyní můžeme nastavit doby trvání jednotlivých oprav a doby mezi příjezdy vozů. Rozdělení pravděpodobností u všech stanovišť oprav je normální. U montáže, demontáže a opravy podvozků nastavíme hodnoty podle tabulky 2. Pro délku opravy sedadel použijeme hodnoty vypočítané podle postupu v předešlém kroku, tedy střední hodnota 1 320 minut a odchylka 312 minut. V zadání stojí, že vstupní proud je Poissonovský se střední hodnotou 2,8 vozu za den, z čehož lze určit, že střední doba mezi příjezdy vozů je přibližně 0,357 dne což odpovídá 514 minutám. Příjezd nového vozu nastavíme tedy na exponenciální rozdělení se střední hodnotou doby před příjezdem dalšího vozu 514 minut.

Na takto nastaveném simulátoru budeme provádět simulační pokusy s různými hodnotami pro kapacitu haly (v zadání jako počet speciálních stolic) a počty pracovníků. Počty pracovníků zde nastavených znamenají počty na jednu směnu, jejich celkový počet je tedy trojnásobný. Délka fronty před halou je v zadání pevně určena na tři vozy. Pro každé nastavení těchto parametrů (scénář) provedeme pět replikací pokusu, u kterých budeme sledovat procentuální počet neuspokojených vozů a procentuální využití podvozkářů a čalouníků. Výsledkem každého běhu budou průměrné hodnoty získané

z těchto pěti replikací. Dobu trvání každého pokusu určíme nastavením maximálního počtu příjezdů na tisíc vozů, což časově odpovídá době zhruba jednoho roku.

3.3.3 Výsledky

V tabulce 3 jsou uvedeny hodnoty získané z běhů simulátoru s různými scénáři. Kompletní tabulka hodnot je v příloze [B]. První scénář představuje nejúspornější možnou konfiguraci, tedy pouze jednu stolici a jednoho pracovníka na směnu od každého druhu. Toto množství obslužných zdrojů je však nedostačující, protože by takto nebyla uspokojena ani polovina vozů přijíždějících do opravny. Proto se v každém dalším scénáři přidává některý ze zdrojů, aby bylo dosaženo lepších výsledků při opravování vozů. Lze sledovat klesající procentuální podíl neuspokojených vozů, tedy rostoucí úspěšnost oprav.

Tab. 3 Získané hodnoty

Číslo scénáře	Počet stolic	Počet podvozkářů	Počet čalouníků	Neuspokojených [%]	Využití podvozkářů [%]	Využití čalouníků [%]
1	1	1	1	**62,94**	23,16	93,618
2	2	1	1	**60,2**	25,03	99,98
3	1	1	2	**31,2**	42,992	85,766
4	2	1	2	**23,94**	48,32	96,878
5	2	1	3	**5,32**	59,752	79,568
6	2	1	4	**2,22**	63,572	63,704
7	2	1	5	**1,64**	62,256	49,948
8	3	1	4	**1,12**	63,14	63,26
9	3	1	5	**0,76**	64,546	51,46
10	2	2	5	**0,28**	30,946	49,872
11	3	2	5	**0,04**	32,264	51,702

Jelikož v zadání není jasně určeno, jakých hodnot chce zadavatel dosáhnout ani jaké jsou náklady na jednotlivé zdroje, nelze jednoznačně určit, který ze simulovaných scénářů je optimální. Z výsledků je však patrné, že scénáře jedna až čtyři jsou z hlediska počtu neuspokojených vozů nevyhovující. Scénáře číslo sedm až jedenáct jsou považovány za úspěšné z hlediska počtu neuspokojených vozů, ovšem z hlediska počtu obslužných zdrojů, tedy i z hlediska nákladů je nevyhovující. Za nejvýhodnější lze považovat scénáře pět a šest, kde se procentuální neúspěšnost pohybuje mírně nad hranicí pěti procent, respektive mírně pod hranicí dvou a půl procenta.

4 Návrh úloh

Součástí této práce je také návrh úloh pro studenty předmětu Modelování a simulace. Jedná se o dvě úlohy, jejichž popis a zadání se nachází níže.

4.1 Úloha 1

4.1.1 Popis

První úloha se zabývá generátory náhodných čísel. Úkolem je doplnit podle daného rozhraní do aplikace simulátoru metody generující pseudonáhodná čísla s daným rozdělením pravděpodobnosti (normální, rovnoměrné, exponenciální) a s danými parametry (střední hodnota, odchylka, minimum, maximum). Rozhraní, ve kterém jsou metody zadány je *IStavSimJadra* a jedná se o metody *dalsiNorm(double stred, double odchylka)* pro normální rozdělení pravděpodobnosti s danou střední hodnotou a odchylkou, *dalsiExp(double stred)* pro exponenciální rozdělení určené střední hodnotou a *dalsiRovn(double min, double max)* pro rozdělení rovnoměrné s danou minimální a maximální hodnotou. Pro připravení úlohy stačí smazat tyto tři metody ve třídě *StavSimJadra,* která dané rozhraní implementuje. Po doplnění může student ověřit správnost vytvořených metod prováděním pokusů na celém simulátoru, nebo pomocí nástroje Input analyzer, který je součásti simulačního programu Arena.

4.1.2 Zadání

V programu chybí generátory pseudonáhodných čísel s různými rozděleními pravděpodobnosti (normální, exponenciální, rovnoměrné). Vaším úkolem je chybějící metody obstarávající toto generování doplnit na základě daného rozhraní. Rozhraní *IStavSimJadra* obsahuje kromě jiného signatury těchto metod.

```
public interface IStavSimJadra {
      ...
      long dalsiNorm(double stred, double odchylka);
      long dalsiExp(double stred);
      long dalsiRovn(double min, double max);
}
```

Metody doplňte do třídy *StavSimJadra*, která toto rozhraní implementuje. Jaké hodnoty a s jakými parametry má která metoda generovat je z jejich názvů patrné. Všechny metody vrací hodnotu typu long, se kterou program dále pracuje při plánování událostí.

Správnou funkčnost vytvořených metod můžete ověřit prováděním pokusů se simulátorem nebo pomocí nástroje Input analyzer, který je součástí simulačního programu Arena.

4.2 Úloha 2

4.2.1 Popis

V této úloze se využívá vytvořená aplikace k účelu, ke kterému je určena tedy k simulaci opravny vozů. Jsou zadány doby jednotlivých oprav a maximální kapacita haly. Úkolem pro studenty je pomocí série simulačních pokusů s tímto simulátorem stanovit optimální množství obslužných zdrojů (podvozkář, dělník) a délku fronty před halou. Výsledky získané ze simulace podle zadání jsou uvedeny v tabulce 4. Délka simulace byla nastavena na 2000 příjezdů, což časově odpovídá zhruba jednomu roku. Podle nich je možná kontrola správnosti řešení.

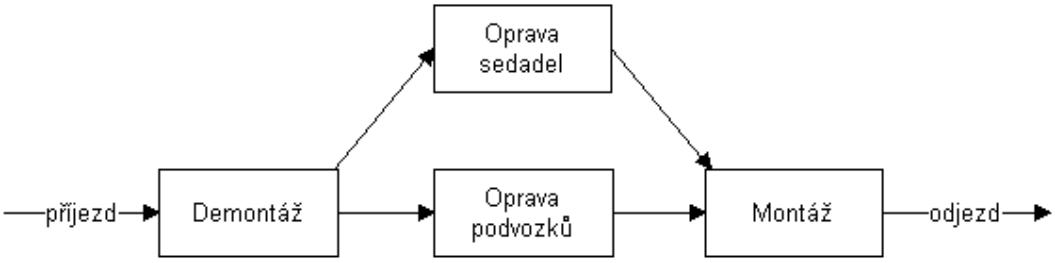
Tab. 4 Získané hodnoty

Číslo scénáře	Délka koleje	Počet podvozkářů	Počet čalouníků	Neuspokojených [%]	Využití podvozkářů [%]	Využití čalouníků [%]
1	0	1	1	**61,09**	50,558	93,122
2	1	1	1	**57,78**	53,35	98,572
3	0	1	2	**42,84**	71,506	66,086
4	1	1	2	**33,11**	85,214	79,582
5	2	1	2	**27,67**	92,3	84,858
6	2	2	2	**22,68**	50,152	92,4
7	2	2	3	**6,8**	59,556	73,236
8	2	2	4	**3,89**	60,526	55,866
9	3	2	3	**3,66**	60,024	73,886
10	2	3	4	**2,06**	41,37	57,118
11	3	2	4	**1,48**	61,132	56,37
12	3	3	4	**0,79**	42,326	58,56

V zadání je požadována 95% úspěšnost, což nejlépe splňují scénáře 8 a 9. Který z nich je vhodnější nelze jasně určit, protože v zadání nejsou uvedeny náklady na jednotlivé pracovníky, ani náklady na výstavbu čekárny před halou.

4.2.2 Zadání

Vzniká nová opravna železničních vozů a Vaším úkolem je stanovit optimální počty pracovníků v ní pracujících a velikost čekárny, do které se vozy před halou opravny řadí jako do fronty. Zmíněná čekárna před vjezdem do haly ještě není vybudovaná a k jejímu vybudování dojde, pouze pokud výsledky simulace ukážou, že se s jejím použitím sníží procento vozů, které nebude možné z důvodu nedostatku místa v hale opravit. S velikostí čekárny samozřejmě poroste i její cena. Prostor před halou je však omezený, proto vybudovaná čekárna může pojmout maximálně tři vozy. Na obrázku je znázorněn postup opravy uvnitř haly.



Obr. 17 Schéma opravny

Doby potřebné k realizování oprav na jednotlivých stanovištích a zdroje k nim potřebné jsou uvedeny v tabulce 5. U všech stanovišť je normální rozdělení pravděpodobnosti. Dělníci pracují ve třísměnném provozu (pracovní doba 8 hodin). Předpokládá se exponenciální rozdělení přijíždějících vozů se střední hodnotou 5,6 vozů za den.

Tab. 5 Doby obsluhy

Název	μ	σ	Zdroj
demontáž	25 min.	5 min.	dělník-podvozkář
oprava sedadel	10 hod.	2 hod.	dělník-podvozkář
oprava podvozků	4 hod.	45 min.	dělník-čalouník
montáž	1 hod.	15 min.	dělník-čalouník

Pomocí aplikace Simulátor opravny železničních vozů proveďte sérii simulačních pokusů a zjistěte optimální počet pracovníků (podvozkářů a čalouníků). Dále zjistěte, zda je vhodné vystavět před halou čekárnu, případně s jakou kapacitou. Počet vozů, které nebude možno opravit, by neměl přesáhnout hodnotu 5 %, aby byla opravna považována za úspěšnou.

5 Závěr

Vytvořená práce ukazuje možnou realizaci simulačního jádra založeného na metodě plánování diskrétních událostí. Jeho funkčnost byla ověřena praktickou ukázkou na příkladu s opravnou železničních vozů. Používání vytvořené demonstrační aplikace není nikterak složité a po přečtení celé práce by její obsluhu měl zvládnout i ten, který se s problematikou simulace nikdy dříve nesetkal. Pohled na zdrojový kód může přiblížit jeden z možných přístupů k realizaci simulačního jádra. Orientaci v něm usnadňuje přiložený diagram tříd.

Součástí práce jsou také dvě úlohy navržené pro studenty předmětu modelování a simulace. Tyto úlohy můžou sloužit jako semestrální práce, či jako příklady k procvičení.

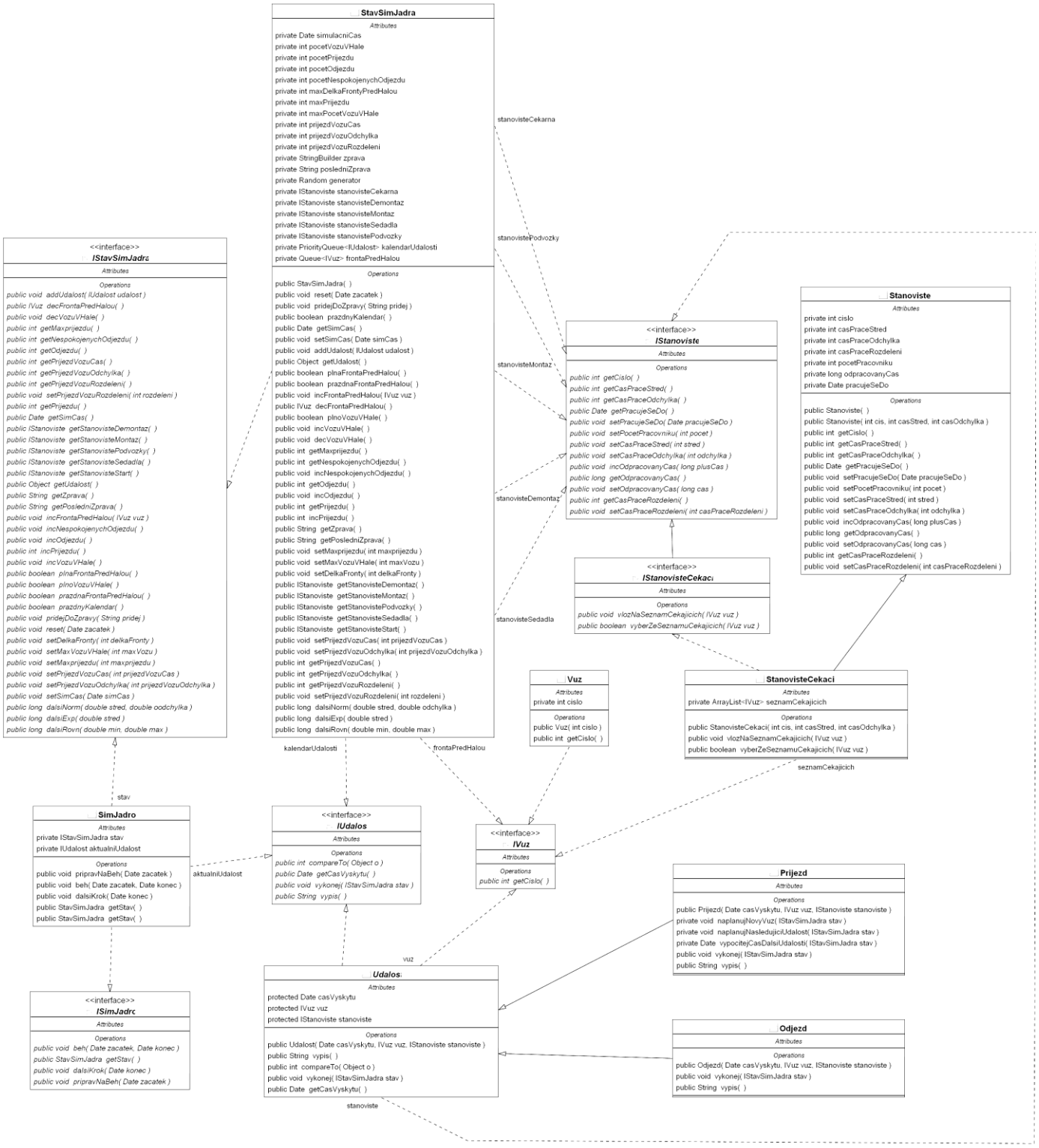
Domnívám se, že prací byly splněny všechny požadavky v jejím zadání a že může být přínosná nejen pro studenty, ale pro všechny, kteří se hodlají zabývat problematikou modelování a simulace.

Literatura

1. **BANKS, J., a další.** *Discrete-event system simulation.* 5. přepracované vydání. New Jersey : Prentice hall, 2009. 622 s. ISBN 0-13-606212-1.
2. **BAŽANT, M.** *Zadání semestrální práce B.* [elektronické zadání pro předměty IMOSI, INSIE 2009/2010] Pardubice : Univerzita Pardubice, 2009.
3. **KAVIČKA, A.** *Datové struktury.* [elektronické sylaby] Pardubice : Univerzita Pardubice, 2009.
4. **KAVIČKA, A.** *Modelování a simulace.* [elektronické sylaby] Pardubice : Univerzita Pardubice, 2009.
5. **KŘÍVÝ, I. a KINDLER, E.** *Simulace a modelování.* [elektronická skripta] Ostrava : Univerzita Ostrava, 2001.
6. **MCLAUGHLIN, B. D., POLLICE, G. a WEST, D.** *Head firts object-oriented analysis & design.* Sebastopol (USA) : O'Reilly, 2007. 600 s. ISBN 0-596-00867-8.

Přílohy

A. Diagram tříd



B. Tabulka hodnot z řešeného příkladu

číslo měření	kapacita haly	počet podvoz.	počet čalouníků	1	2	3	4	5	průměr
1	1	1	1	**64,8** 23,12 93,7	**64** 23,36 93,5	**62,7** 23,34 93,69	**60,8** 22,71 93,61	**62,4** 23,27 93,59	**62,94** 23,16 93,618
2	2	1	1	**60,2** 24,89 99,98	**59,9** 24,8 99,98	**61** 25,22 99,98	**59,9** 25,23 99,98	**60** 25,01 99,98	**60,2** 25,03 99,98
3	1	1	2	**30,7** 42,62 86,37	**31** 42,99 85,82	**29,9** 42,25 84,25	**32,2** 44,35 86,88	**32,2** 42,75 85,51	**31,2** 42,992 85,766
4	2	1	2	**27** 48,5 97,37	**22,4** 46,99 95,21	**23,8** 48,31 97,07	**22,9** 49,1 97,88	**23,6** 48,7 96,86	**23,94** 48,32 96,878
5	2	1	3	**5,5** 59,37 79,18	**5,1** 58,58 78,7	**4,6** 61,83 81,89	**6** 58,93 78,97	**5,4** 60,05 79,1	**5,32** 59,752 79,568
6	2	1	4	**1,9** 65,39 66,35	**2,4** 61,4 61,23	**1,2** 62,5 62,2	**2,1** 62,55 62,95	**3,5** 66,02 65,79	**2,22** 63,572 63,704
7	2	1	5	**1,8** 61,71 49,57	**1,9** 64,56 52,21	**1,6** 61,1 48,86	**0,9** 60,16 47,75	**2** 63,75 51,35	**1,64** 62,256 49,948
8	3	1	4	**2** 64,07 64,64	**1,2** 63,5 63,3	**0,3** 63,07 63,63	**1** 62,91 62,44	**1,1** 62,15 62,29	**1,12** 63,14 63,26
9	3	1	5	**0,5** 64,06 51,25	**1** 64,02 51,02	**0,8** 64,21 50,89	**1,3** 67,19 53,66	**0,2** 63,25 50,48	**0,76** 64,546 51,46
10	2	2	5	**0,1** 30,49 49,54	**0,2** 32,1 51,77	**0,6** 30,94 49,53	**0** 30,85 49,59	**0,5** 30,35 48,93	**0,28** 30,946 49,872
11	3	2	5	**0** 32,32 51,78	**0,2** 31,91 51,08	**0** 31,48 50,84	**0** 32,93 52,84	**0** 32,68 51,97	**0,04** 32,264 51,702

neuspokojených [%]
využití podvozkářů [%] využití čaouníků [%]

Přiložené CD

Na přiloženém CD se nachází vytvořená aplikace *Simulátor opravny železničních vozů*, včetně zdrojových kódů a diagramu tříd.