Slovenská technická univerzita

Fakulta informatiky a informačných technológií
Ústav informatiky a softvérového inžinierstva
Ilkovičova 3, 812 19 Bratislava

Modelovanie a riadenie systému automaticky navádzaných vozidiel pre dopravu vo výrobných procesoch

Študijný odbor : Počítačové systémy a siete
Ročník : 1. inž.
Dátum odovzdania : 20.5.2008

Vedúci tímu : Ing. Jana Flochová, PhD.
Tím č.5 : Bc. Peter Jakubis, Bc. Eva Danillová, Bc. Radoslav Katreniak, Bc. Lukáš Böhm, Mgr. Peter Šúň

0 Obsah

1 Úvod

V tejto časti je uvedený úvod k riešenému projektu. Je tu uvedený cieľ projektu, zadanie projektu a motivácia členov tímu pre riešenie daného problému. Ďalej nasleduje prehľad dokumentu.

1.1 Cieľ projektu

Cieľom tohto projektu je vyriešiť pridelené zadanie v rámci predmetu Tímový projekt. Názov projektu je „Modelovanie a riadenie systému automaticky navádzaných vozidiel pre dopravu vo výrobných procesoch“ a bol zadaný Fakultou informatiky a informačných technológií STU v Bratislave. Pedagogickým vedúcim projektu je Ing. Jana Flochová, PhD. Členmi tímu sú piati študenti inžinierskeho štúdia FIIT STU (Bc. Peter Jakubis, Bc. Eva Danillová, Bc. Radoslav Katreniak, Bc. Lukáš Böhm a Mgr. Peter Šúň).
Na projekte sa pracuje dva semestre a tím má k dispozícii v každom týždni semestra jedno stretnutie s pedagogickým vedúcim. Taktiež je potrebná ďalšia spoločná práca členov tímu mimo stretnutí.

1.2 Zadanie projektu

Systém automaticky navádzaných vozidiel (automatic guided vehicles – AGV) je dôležitou súčasťou komplexne automatizovaných výrobných procesov. Bezkolízne a optimalizované plnenie dopravných úloh napomáha ku zefektívneniu výroby.

V rámci tímového projektu riešte nasledujúce úlohy:

- analyzujte a vyhodnoťte dopravné systémy používané vo výrobe z funkčného hľadiska
- naštudujte a urobte rozbor modelovacích prostriedkov, ako sú Petriho siete, časové okná a iné, z hľadiska ich vhodnosti pre triedu zónovo riadených dopravných prostriedkov

- pre vybranú triedu dopravných systémov spracujte metódu ich modelovania zvolenými prostriedkami

- navrhnite riadenie pre riešenú triedu dopravných systémov

- pre modelovanie a riadenie riešenej triedy dopravných systémov vypracujte a odskúšajte programový systém na modelovanie a riadenie.

1.3 Motivácia

Systém automaticky navádzaných vozidiel je zaujímavý pre riešenie dopravných kolízií, urýchlenie dopravy, a správy ciest. Je to téma ktorá sa stala hlavným bodom nášho projektu. Existuje niekoľko spôsobov ako sa dajú tieto dopravné situácie odsimulovať. Niekoľkým z nich sa v úvode budeme stručne venovať a popíšeme ich správanie a zápis. V podstate sa jedná o rôzne formy jazykov, ktoré nám umožňujú popísať správanie a štruktúru systému, kt. sa budeme snažiť navrhnúť a odsimulovať. Zaujímavou časťou projektu budú simulácie, kde musíme dokázať, že systém je navrhnutý správne a nedôjde ku kolízií, alebo kolíziu systém bude nejakým spôsobom musieť vyriešiť sám a optimalizovať celú premávku, alebo resp. chod systému, aby to nemalo výrazný dopad na jednotlivé úseky, kt. by mohli byť možnou kolíziou zastavené.

Motivujúce je na tomto projekte pre nás všetkých to, že sú oblasti v ktorých tieto poznatky aj prakticky využijeme. V doprave by takéto riešenia veľmi radi určite uvítali, avšak vzhľadom na ich zložitosť a nasedenie do terénu sa používajú jednoduché časovače, semafory príp. niekde je nutné nasadenie ľudského faktora. Budeme sa snažiť navrhnúť systém aby sa nikdy nezastavil, čo v reály však nie je možné, lebo vplyv rôznych neočakávaných faktorov prispieva k tomu že čo sa pokaziť môže, to sa pokazí.

1.4 Prehľad dokumentu

Projektová dokumentácia je členená do 7mich kapitol (častí). V kapitolách sú opísané jednotlivé kroky spracovania projektu začínajúc opisom problémovej oblasti, ďalej jeho analýzou a špecifikáciou, hrubým návrhom, návrhom riešenia, implementáciou tohto projektu končiac.

Prvá kapitola obsahuje úvod k projektu. Je tu uvedený cieľ projektu, zadanie projektu a motivácia členov tímu pre riešenie tohto projektu. Ďalej je tu možné nájsť prehľad dokumentu a slovník používaných pojmov.
Druhá kapitola obsahuje opis problémovej oblasti.
V tretej kapitole je podrobne rozpracovaná analýza súčasných systémov, analýza procesov a dátového modelu.
Štvrtá kapitola obsahuje špecifikáciu požiadaviek na používateľské rozhranie a možné námety na vylepšenia.
V rámci piatej kapitoly je vypracovaný hrubý návrh harmonogramu, návrh nového dátového modelu, návrh tlače a používateľského rozhrania.
V šiestej kapitole je popísaná implementácia nášho projektu, návrh štruktúry siete, na ktorej simulujeme pohyb dopravného systému, využité algoritmy a takisto sú popísané deadlocky, kolízie a ich riešenia.
V poslednej siedmej kapitole sa nachádza záver a zhodnotenie.

1.5 Slovník pojmov

- **PN** – Petriho sieť (Petri Net)
- **P/T** – Petriho sieť
- **CPN** – Farebná Petriho sieť (Coloured Petri Net)
- **AGV** – automaticky navádzané vozidlá (automatic guided vehicles)
- **help** – návod

2 Opis riešeného problému

Rôzne druhy modelovacích jazykov sú uplatňované v rôznych disciplínach, ako veda o počítačoch, manažment informácií, modelovanie biznis procesov, vývoj software a vývoj systémov. Modelovacie jazyky môžu byť použité na špecifikovanie požiadaviek, štruktúry a správania. Mali by byť používané na presné určenie systému, tak aby bolo ľahšie porozumieť fungovaniu modelovaného systému. Viac vyzreté modelovacie jazyky sú presné, konzistentné a vykonateľné. Základné techniky vytvárania diagramov by mali umožniť vytvoriť obrazovú reprezentáciu požiadaviek, štruktúry a správania systému. Vykonateľné modelovacie jazyky s použitím správnych nástrojov ponúkajú automatizáciu verifikácie, validácie, simulácie systému a generovanie kódu z obrázkovej reprezentácie.

AGV pomáhajú znížiť náklady a zvyšujú efektivitu výrobných systémov. Odvetvia používajúce AGV musia mať nad ich riadením nejaký druh kontroly. Ak majú k dispozícií funkčný model riadenia, je pre nich jednoduchšie riešenie problémov, ktoré môžu nastať ako aj samotné sledovanie každého jedného vozidla.

Problémom oblasti modelovania a simulácie dopravných systémov pre AGV je práve návrh modelu systému riadenia vozidiel v dopravnom systéme čo najefektívnejším spôsobom. Model riadenia treba navrhnúť tak aby časová strata čakania na uvoľnenie trasy bola pre každý prípad čo najmenšia, samozrejmosťou je úplné odstránenie možnosti kolízie dvoch vozidiel.

3 Analýza

Vo fáze analýzy projektu a dostupných riešení sme sa rozhodli túto časť rozdeliť do nasledovných okruhov: analýza vybraných modelovacích jazykov a analýza existujúcich riešení pre modelovanie a simuláciu.

3.1 Analýza vybraných modelovacích jazykov

Pre analýzu sme vybrali tieto dva modelovacie jazyky: Petriho siete a stavové automaty.

3.1.1 Petriho siete

3.1.1.1 Petriho siete

Petriho siete sú matematický a grafický nástroj na modelovanie a analýzu systémov a diskrétnych udalostí. Najmä sú vhodné na modelovanie diskrétnych udalostí obsahujúcich paralelizmus, zdieľanie zdrojov, väzbu na reálny čas a väzbu na okolie. Petriho siete vznikli rozšírením modelovacích možností konečných automatov. Umožňujú veľmi efektívne popisovať informačné toky v modelovanom systéme.

Použitie našli v aplikáciách súvisiacich s akýmikoľvek paralelnými štruktúrami, od počítačových systémov cez automatizované systémy riadenia, dopravné systémy, telekomunikačné systémy až po použitie v administratíve a v ďalších oblastiach.
Sú pomenované podľa nemeckého matematika Carla Adama Petriho, ktorý v roku 1962 ako prvý použil tento model správania sa systémov, ich vzájomných vzťahov a subsystémov.

Praktické aplikácie - najčastejšie dva prístupy:

1) Petriho siete ako pomocný prostriedok na analýzu návrhu - prostriedok modelovania

2) Petriho siete ako návrhový a špecifikačný prostriedok - implementačný prostriedok

3.2 V oboch prípadoch - prostriedok na zápis, analýzu prípadne simuláciu Petriho siete, ktorý určí vlastnosti modelu (DesignCPN, Pipe, Pesim).

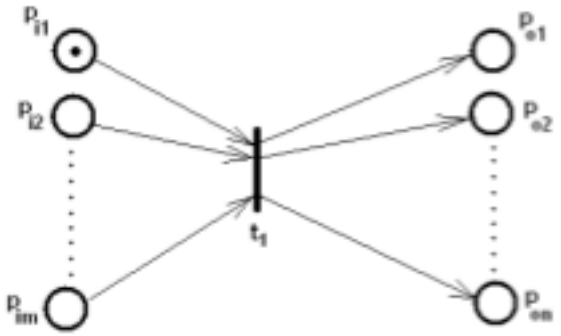
Podstatnými výhodami použitia Petriho sietí pre riadenie priemyselných systémov sú napríklad:

- Možnosť kontroly systému na nežiaduce vlastnosti ako napríklad existencia mŕtvych stavov, neohraničenosť
- Jednoduchosť modelovania systémov, majú pomerne jednoduchú a názornú grafickú reprezentáciu.
- Možnosť vykonávať analýzu výkonu systému a jeho simuláciu
- Možnosť generovať supervízorove riadenie priamo z Petriho siete

Nevýhodou je, že modelovaním niektorých systémov pomocou Petriho sietí môže vzniknúť rozsiahla Petriho sieť, ktorá sa potom náročne analyzuje.

3.2.1.1 Základné definície Petriho sietí

Z hľadiska teórie grafov je Petriho sieť orientovaný bipartitný ohodnotený jednoduchý graf . Základnými stavebnými prvkami Petriho siete sú krúžky, ktoré sa nazývajú *miesta* a zvislé neorientované úsečky, ktoré sa nazývajú *prechody*, ktoré sú zviazané s udalosťami v systéme. Prechody a miesta sa spájajú do grafu pomocou orientovaných úsečiek nazývaných *hrany.* $p_{i_1}, p_{i_2}, ..., p_{i_m}$ Miesta v Petriho sieti reprezentujú $p_{o_1}, p_{o_2}, ..., p_{o_n}$ stavy, alebo podmienky. Ak sa v mieste nachádza značka, znamená to, že podmienka je splnená. Na obrázku 1 je základná konštrukcia Petriho siete. Miesta , sú vstupné miesta prechodu t_1 a miesta
sa nazývajú výstupné miesta prechodu t_1.



Obrázok 1 - Petriho sieť

3.2.1.2 Definícia Petriho siete

Petriho sieť je definovaná ako 5-tica $N = (P,T,F,W,M_0)$, kde

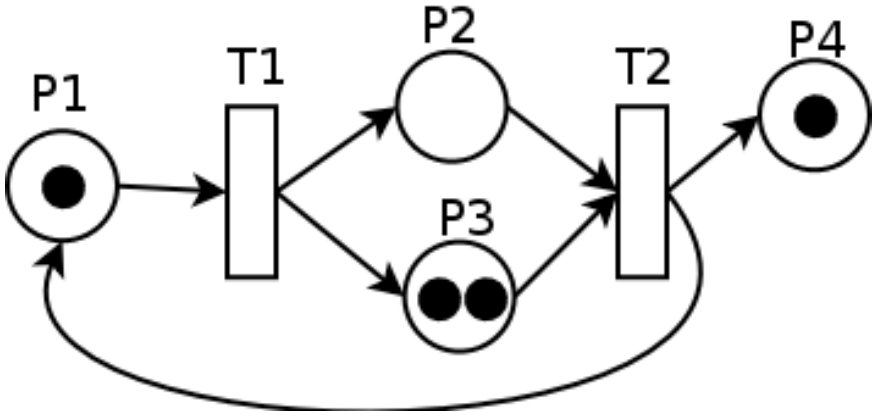
- P a T sú neprázdne konečné množiny označujúce P (places) – miesta a T (transitions) – prechody pričom platí $P \cap T = C$
- F je teda množina usporiadaných dvojíc pozostávajúcich z miesta a prechodu, alebo prechodu a miesta a nazýva sa aj toková relácia. Platí, že a platí, že

$$\forall p_i \in P : \exists t_j \in T \text{ také, že } (p_i, t_j) \in F \text{ a}$$
$$\forall t_s \in T : \exists p_r \in P \text{ také, že } (t_{si}, p_{rj}) \in F$$

  - Inými slovami možno povedať, že každé miesto je najmenej v jednej z usporiadaných dvojíc v F a každý prechod sa vyskytuje v F aspoň raz. Teda v Petriho sieti nie sú žiadne izolované miesta ani prechody.
- W je váhová funkcia definovaná ako W:F→ I *kde I* je množina kladných konečných čísel. Čísla, ktoré sú touto funkciou priradené F nazývame váhy. Znamená to, že každá hrana spájajúca prechod a miesto má priradené číslo, ktoré zodpovedá počtu značiek, ktoré sa pri vykonaní prechodu odoberú z jeho vstupného miesta, alebo sa pridajú do jeho výstupného miesta. Ak hrana nemá uvedené číslo, potom sa implicitne predpokladá váha rovná jednej .
- M_0 je funkcia určujúca počiatočné označkovanie Petriho siete: $M_0 : p \to I^+ \cup \{0\}$, kde I^+ je množina kladných konečných čísiel.

Je nutné však podotknúť, že existuje viacero formálnych definícií.

Príklad Petriho siete :



Obrázok 2 - Príklad Petriho siete

3.2.1.3 Ohraničenosť

Petriho sieť je *k-ohraničená* (alebo iba *ohraničená)* ak počet značiek v každom mieste siete neprekročí konečný počet *k* pre ľubovoľné značenie dosiahnuteľné z *Mo,* t.j. $M(p) \leq k$ pre každé miesto p a každé dosiahnuteľné značenie M_1, $\in R(N, M_O)$.

3.2.1.4 Bezpečnosť

Sieť *N* je *bezpečná* (tiež binárna) ak /c = 1, teda ak počet značiek v každom mieste môže byť maximálne jednotkový. Bezpečnosť siete si môžeme „vynútiť" ak použijeme Petriho sieť s kapacitami, pričom každému miestu priradíme kapacitu 1. „Prirodzená" bezpečnosť Petriho siete však priamo súvisí so štruktúrou siete a s počiatočným značením.

3.2.1.5 Pokryteľnosť

Pre neohraničenú Petriho sieť je možné konečným spôsobom reprezentovať množinu značení prostredníctvom vlastnosti pokrytia.

Značenie *M* z (N,M_0) je *pokryteľné,* ak existuje značenie *M'* (z množiny značení dosiahnuteľných zo značenia M_0) také, že $M' > M$.

3.2.1.6 Živosť

Petriho sieť N je *živá,* ak je možné z ľubovoľného značenia M dosiahnuteľného z M_0 spustiť ľubovoľný prechod siete PN nejakou postupnosťou spúšťania.

(1) Prechod $t \in T$ je živý na hladine 0, ak nie je spustiteľný pre žiadne značenie M_0 (t.j. je neživý)

(2) Prechod $t \in T$ je živý na hladine 1, ak existuje také $M \in M_0 >$ také, že prechod ŕ je M-spustiteľný.

(3) Prechod $t \in T$ je živý na hladine 2, ak pre každé n ∈ N existuje výpočtová postupnosť Petriho siete taká, že sa v nej prechod t vyskytuje aspoň n-krát.

(4) Prechod $t \in T$ je živý na hladine 3, ak existuje výpočtová postupnosť Petriho siete taká, že sa v nej prechod t vyskytuje nekonečne veľa krát.

(5) Prechod $t \in T$ je živý na hladine 4, ak pre každé $M \in M_0 >$ existuje také značenie M', že M' $\in [M>$ a že prechod í je M'-spustiteľný.

Petriho sieť je živá na hladine *h,* H∈{0,1,2,3,4}, ak pre každý prechod t∈T je živý na hladine *h.*

3.2.1.7 Bezkonfliktnosť

Sieť (*N,Mo)* je *bezkonfliktná,* ak pre ľubovoľné dva spustiteľné prechody spustenie jedného z nich neovplyvní spustiteľnosť druhého. Ak je prechod v bezkonfliktnej sieti spustiteľný, ostáva spustiteľný až do okamihu svojho spustenia.

3.2.1.8 Konzervatívnosť

Sieť (N,M_0) je striktne *konzervatívna,* ak pre ľubovoľné značenie M dosiahnuteľné z M_0 ostáva celkový súčet značiek zo všetkých miest siete konštantný.

3.2.1.9 Časované Petriho siete

Časované Petriho siete sa delia na deterministické a stochastické Petriho siete. V dokumente sa budeme venovať iba deterministickým časovaným sieťam.

Deterministické časované siete umožnujú popísať systém závislý od času. Berú do úvahy skutočnosť, že medzi začiatkom a koncom operácie prejde určitý čas.
Používajú sa dve základné metódy modelovania:

- Časovanie priradeného stavu – P – časovanie
- Časovanie priradeného prechodu – T – časovanie

P – časované PS :

Sú definované usporiadanou šesticou pričom prvých päť parametrov je zhodných ako pri normálnych PS a šiesty parameter *Tempo* predstavuje zobrazenie množiny stavov *P* do množiny reálnych čísiel, ktoré sú kladné, alebo nula. $Tempo(P_i) = d_i$ je interval času priradený stavu P_i .
V našom prípade, po príchode vozíka do miesta P_i sa tento stáva nedostupným aspoň po dobu d_i .

T – časované PS:

Sú taktiež definované usporiadanou šesticou pričom prvých päť parametrov je zhodných ako pri normálnych PS a šiesty parameter Tempo je zobrazenie množiny *T* do množiny reálnych čísiel, ktoré sú kladné, alebo nula. $Tempo(T_i) = d_i$ je interval času priradený prechodu T_i .
V našom prípade, prechod môže byť v dvoch stavoch, buď rezervovaný pre prepálenie prechodu T_i, alebo nerezervovaný.

3.2.1.10 Plusy a mínusy Petriho sietí

Plusy Pretiho sietí:

- umožňujú simulácie možných stavov
- umožňujú zistiť kolízne stavy

- na základe vytvoreného modelu nám uľahčujú lepšie pochopenie celého procesu

Mínusy Petriho sietí:

- pri rozsiahlych systémoch môžu vzniknúť veľmi veľké Petriho siete, ktoré môžu byť neprehľadné, a ich ďalšia analýza býva komplikovaná.

3.2.1.11 Diagnostika porúch na základe analýzy stromu dosiahnuteľnosti

Diagnostika porúch na základe analýzy stromu dosiahnuteľnosti je často krát náročná úloha, pokiaľ sa jedná o veľké a distribuované systémy. Je založená na spätnej analýze stromu dosiahnuteľnosti a na metóde spätného spúšťania prechodov Petriho sietí (backward firing Petri nets). Cieľom je mať model ohraničený, reverzibilný a živý.

Zo stromu dosiahnuteľnosti sme schopní zistiť množinu dosiahnuteľných stavov. Zakázaním určitých hrán, t.j. udalostí môžeme zabezpečiť dosiahnuteľnost' určitých stavov a metódou spätného spúšťania prechodov Petriho sietí, t.j. analýzou minulosti dokážeme zistiť, s ktorých stavov sme sa mohli dostať do stavu, v ktorom sa nachádzame. Pri danej incidenčnej matice a počiatočnom značení Petriho siete sme schopní zistiť, čo sa stane v budúcnosti, alebo aj možné predchádzajúce stavy -to čo sa stalo v minulosti. Pri programovej realizácii postupujeme obdobným spôsobom, ako pri algoritme konštrukcie stromu dosiahnuteľnosti, ale nepočítame nasledujúce stavy a sekvenčné udalosti, ale predchádzajúce stavy pomocou stavovej rovnice Petriho siete.

3.2.1.12 Diagnostika porúch na základe analýzy stromu dosiahnuteľnosti reverznej Petriho siete

Na základe tejto metódy dokážeme spätne analyzovať a určiť predchádzajúce stavy Petriho siete, ktorú prevedieme na reverznú a spätným spúšťaním prechodov vypočítame strom dosiahnuteľnosti.

Výpočet stromu dosiahnuteľnosti realizujeme nasledovným spôsobom:

Danú Petriho sieť *N* prevedieme na reverznú Petriho sieť *N* otočením orientácie hrán, a to pokiaľ vedie hrana z miesta p na prechod t v sieti *N,* potom hrana bude mať otočenú orientáciu z prechodu *t* na miesto p v sieti *N* a naopak.

3.2.2 Stavové automaty

Teória formálnych jazykov a automatov predstavuje veľmi dôležitú oblasť informatiky. Základy novodobej histórie tejto disciplíny položil v roku 1956 americký matematik Noam Chomsky, kt. vytvoril matematický model gramatiky jazyka. Realizácia pôvodných predstáv, formalizovať popis prirodzeného jazyka takým spôsobom, aby mohol byť preklad z jedného prirodzeného jazyka do druhého automatizovaný, alebo aby prir. Jazyk slúžil ako prostriedok komunikácie človeka s počítačom sa stal ukázal ako veľmi ťažký a ani súčasné výsledky nie sú uspokojujúce.

Začala sa vyvíjať vlastná teória jazykov, ktorá pracuje s dvoma duálnymi matematickými entitami s gramatikou a automatom, prestavujúci abstraktný matematický nástroj. Pričom gramatika umožňuje popísať štruktúru viet formálneho jazyka, automat dokáže túto štruktúru identifikovať.

3.2.2.1 Základné pojmy a definície

Abeceda Abeceda Σ je konečná množina symbolov.

Reťazec (slovo) α prvkov z konečnej množiny Σ je ľubovoľná konečná postupnosť prvkov tejto množiny. Reťazce spravidla označujeme gréckymi písmenami. Počet prvkov v reťazci udáva jeho dĺžku a označujeme ju |α|. Reťazec, ktorý neobsahuje žiadny prvok, nazývame prázdny reťazec a označujeme ho ε alebo e (pokiaľ nepríde k zámene). Jeho dĺžka je 0.
α = a1a2...ak-1ak ai□Σ, i = 1,2,...,k; |α| = k

3.2.2.2 Konečný automat

Ak hovoríme o jazyku ako množine slov a gramatike ako o súbore pravidiel, ktoré umožňujú generovať slová z jazyka, potom automat je prostriedkom ako zaistiť, ktoré slová

do jazyka patria a ktoré tam nepatria. Automat nemusí mať v určitom kroku definované do akého stavu má prejsť alebo môže mať viac možností, potom hovoríme, že je automat nedeterministický.

Konečný automat (deterministický) – KA

(Deterministickým) konečným automatom (DKA) nazývame každou päticu A =(Q, Σ, δ, q0, F), kde:

- Q je konečná neprázdna množina (*množina stavov, stavový priestor*)
- Σ konečná neprázdna množina (*množina vstupných symbolov, vstupná abeceda*)
- δ je zobrazenie Q ×Σ → Q (*prechodová funkcia*)
- q0 □ Q (*počiatočný stav, iniciálny stav*)
- F □ Q (*množina koncových stavov, cieľová množina*).

***Jazykom rozpoznávaným konečným automatom A*, potom nazveme**

množinu L(A)={ w|w □ Σ* □ δ*(q0,w) □ F }.

Povedzme, že *jazyk* L (nad abecedou Σ) *je rozpoznateľný konečným automatom* , ak existuje konečný automat *A je* taký, že L(A)=L.

Rozpoznávaný jazyk je jazyk rozpoznávaný konkrétnym automatom – ide o slová ktoré ho dostanú do koncového stavu.

Rozpoznateľný jazyk je taký, pokiaľ je k nemu vôbec možné zostrojiť KA.

Možnosti reprezentácie konečného automatu:

- Zápis jednotlivých prvkov pätice konečného automatu
- Tabuľka
- Stavový diagram
- Stavový strom

Majme konečný automat A=(Q, Σ, δ, q0, F)

Q={q0,q1,q2,q3}, Σ = {0,1}

δ(q0,0)=q0, δ(q0,1)=q2

δ(q1,0)=q1, δ(q1,1)=q3

δ(q2,0)=q1, δ(q2,1)=q3

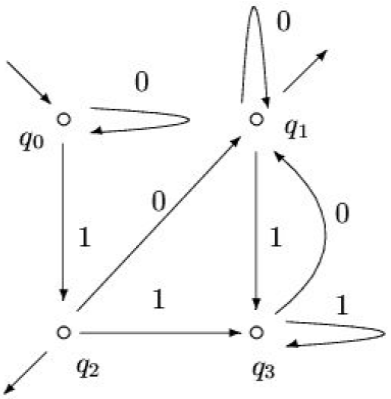
δ(q3,0)=q1, δ(q3,1)=q3

F={q1,q2}

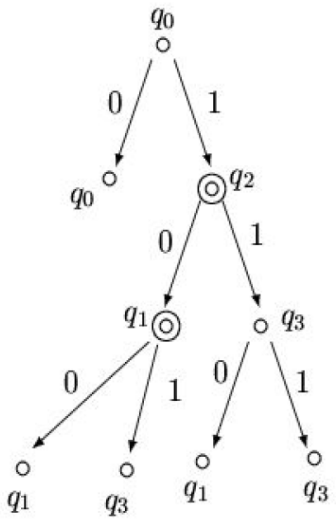
TABULKA:

Q\Σ	0	1
→ q0	q0	q2
← q1	q1	q3
← q2	q1	q3
q3	q1	q3

STAVOVÝ DIAGRAM:



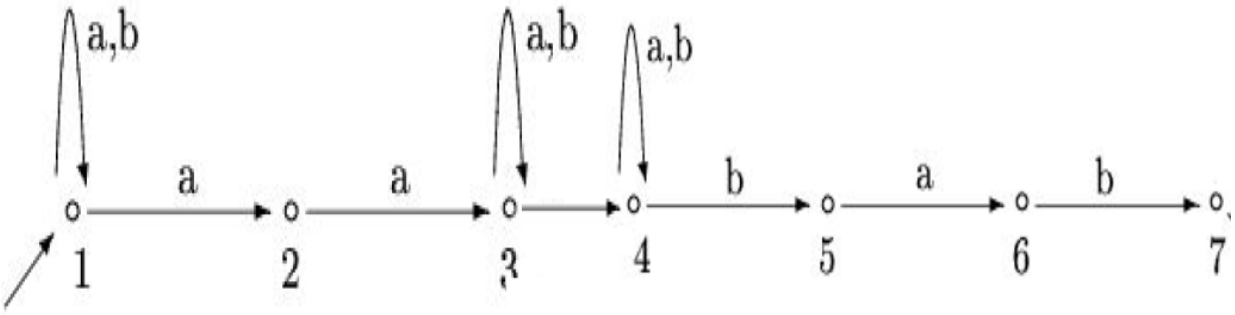
STAVOVÝ STROM:



Konečný automat (nedeterministický)

Nedeterministickým konečným automatom (NKA) budeme nazývať päticu A =(Q, Σ, δ, I, F) , kde:

- Q a Σ sú po rade neprázdne množiny stavov a vstupných symbolov,
- δ:Q×Σ → P(Q) je prechodová funkcia (P(Q) je množina všetkých podmnožín množiny Q).
- I □ Q je množina počiatočných stavov a F □ Q je množina koncových stavov.



Nedeterminizmus vyžaduje len to aby riešenie existovalo, neurčuje v akom čase sa má dosiahnuť.

3.3 Analýza existujúcich riešení

V súčasnosti existuje široká ponuka existujúcich softwarových riešení pre rôzne modelovacie jazyky. Odlišujú sa podporovanými modelovacími jazykmi, alebo ich variáciami, spôsobom zobrazenia simulácie, možnosťami nastavenia a tak ďalej.

3.3.1 Petri .NET Simulator

Je to aplikácia pre navrhnutie, nakreslenie a simuláciu Petriho sieti. Je navrhnutý pre modelovanie, analýzu a simuláciu flexibilných výrobných systémov, ale môže byť taktiež použitý pre iné systémy.

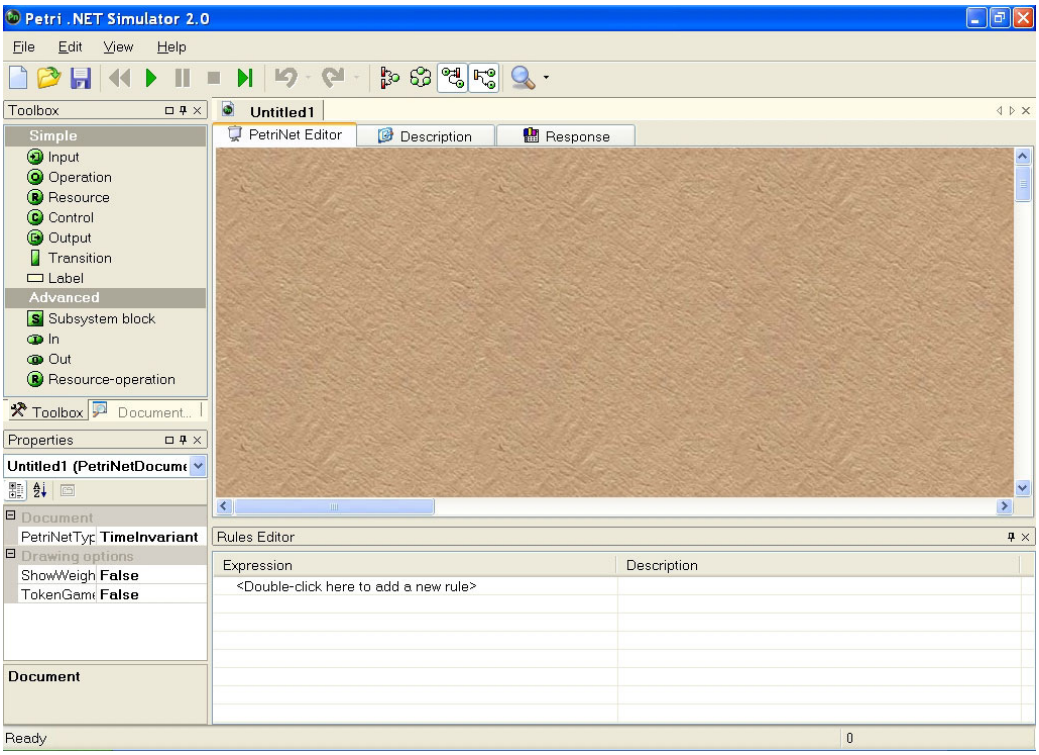
Základné vlastnosti:

- jednoduché kreslenie Petriho sieti. Je možné nakresliť mnoho tvarov PN.
- Objekty PN (miesta a prechody) môžu byť spájané do podsystémov. PN je tak ľahšie porozumieť a udržovať
- Simulácia časových PN
- Simulácia môže zobraziť animácie značiek a čas spracovania
- Simulácia môže byť spustená v reálnom čase, alebo je ju možné zobraziť vo forme tabuľky alebo ako oscilogram. Oscilogram sa dá použiť pri výbere časového intervalu, ktorý bude použitý pre vykonanie analýzy siete
- Použitie kontrolného algoritmu cez editor pravidiel na PN model aby sa vytvoril stabilný systém

- Export modelu PN alebo oscilogramu do zdokonaleného metafile formátu, alebo jednoducho preniesť model do aplikácie tretej strany cez clipboard. Exportovať ako tabuľku, výhodne pri písaní výsledkov simulácie.

Požiadavky:

- x86 kompatibilné PC
- Microsoft® Windows®
- Microsot® .NET Framework 1.1



3.3.2 TimeNET

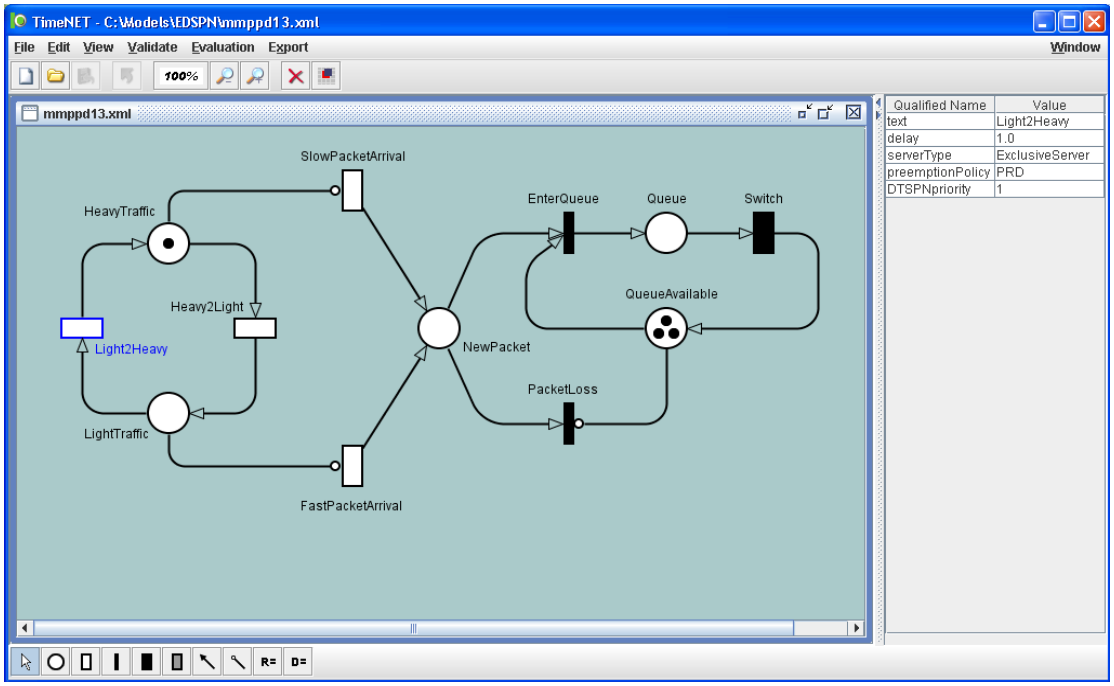
Softwarový balíček TimeNET (verzia 4), grafický a interaktívny nástroj pre modelovanie stochastických PN a stochastických farebných PN. TimeNET je vyvinutý Real-Time Systems a Robotics Technickej Univerzity Berlin. Projekt bol motivovaný potrebou silného softvéru pre efektívne hodnotenie časovaných Petriho sieti s ľubovoľným časom spúšťania prechodov. TimeNET a jeho predchodca DSPNexpress boli ovplyvnené skúsenosťami s inými dobre známymi nástrojmi pre PN ako GreatSPN a SPNP.

Základné vylepšenia vo verzii 4.0:

- všeobecné grafické rozhranie v prostredí JAVA založené na reprezentácií XML sieťovej triedy, ľahko rozšíriteľné pre väčšinu aplikácií založených na princípe grafu
- užívateľské prostredie a algoritmus zhodnotenia bežia pod operačným systémom Windows® a Linux
- modelovania a simulácia stochastických farebných PN
- grafické zobrazenie výsledku simulácie stochastických farebných PN
- nezávislé komponenty pred modelovanie, simuláciu, analýzu a výsledkov čo umožňuje GUI byť spustené na inom počítači ako modul analýzy.

Požiadavky:

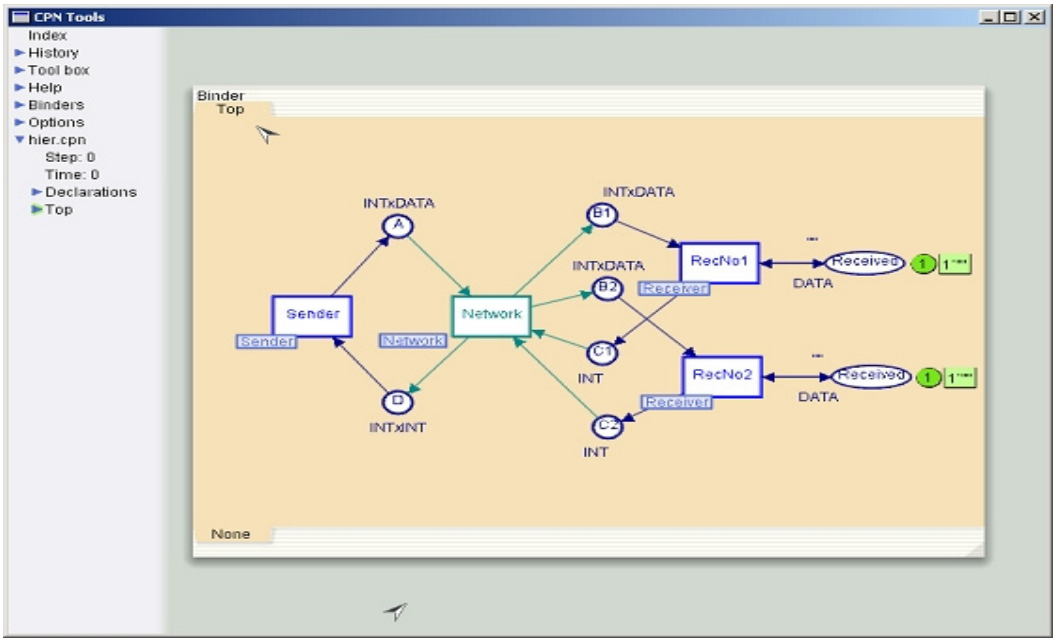
- Windows® XP alebo Linux (odporúčaná distribúcia Debian)
- Java Runtime Environment verzia 5 alebo novšia



TimeNET 4.0

3.3.3 CPN Tools

CPN Tools je nástroj pre editáciu a analýzy farebných PN. GUI je založené na pokročilých interaktívnych technikách. Aplikácia používa pre informovanie používateľa kontextové chybové hlásenia a indikuje závislosti medzi jednotlivými elementmi PN. Nástroj poskytuje kontrolu syntaxe a generácie kódu, čo sa vykonáva počas konštruovania siete. Rýchly simulátor zvládne časované aj nečasovaná PN. Môžu byť vygenerované a analyzované úplné alebo čiastočné vyhodnotenia PN. Štandardné vyhodnotenie obsahuje informácie ako ohraničenie, živosť.



4 Špecifikácia požiadaviek

4.1 Funkčná špecifikácia

Našim hlavným zámerom je vytvorenie komplexného produktu na báze grafickej simulácie modelu riadenia pomocou časovaných Petriho sieti. Bude predstavovať integráciu dvoch hlavných funkcií do jedného a to funkciu nakreslenia modelu a jeho simulácia.

Na základe analýzy podobných riešení, stretnutí s vedúcou tímu a našich vzájomných diskusií sme postupne dospeli k záväzným a spresneným požiadavkám na systém. Tieto požiadavky sme rozdelili do nasledovných kategórii:

- požiadavky na používateľské rozhranie a prácu v programe
- požiadavky na implementovanie elementov
- požiadavky na simuláciu a na použitie

Zoznam všeobecných požiadaviek:

- na prostriedky nenáročná aplikácia
- prenositeľnosť aplikácie
- intuitívne grafické rozhranie
- ľahké nakreslenie modelu pomocou Petriho siete
- editácia už vytvoreného modelu
- simulácia modelu
- vyhodnotenie modelu
- matematické zobrazenie modelu pomocou maticovej reprezentácie
- zobrazenie modelu pomocou binárneho stromu

4.2 Špecifikácia používateľského rozhrania

Na navrhovaný systém sú kladené nasledovné požiadavky, ktoré súvisia s používateľským rozhraním a prácou v programe.

Systém má poskytovať:

- plne grafické používateľské prostredie s prehľadným a jednoduchým ovládaním
- zobrazenie a vytvorenie zoznamu všetkých elementov v PN
- zobrazenie a vytvorenie zoznamu všetkých typov spojenia elementov
- jednoduché vytváranie modelu použitím zoznamu elementov a spojení metódou drag and drop
- jednoduché pridanie / odobratie elementu PN
- uloženie a znovu otvorenie uloženého modelu
- vytvorenie helpu, resp. jednoduchého pomocníka integrovaného do programu
- umožnenie tlače vytvoreného modelu
- jednoduchý prechod do konfiguračnej časti programu

4.3 Špecifikácia implementovania elementov

Miesto:

- meno
- identifikátor
- typ vstupu
- poloha
- značky
- čas (v prípade časovaných PN)
-

Prechody:

- meno
- identifikátor
- poloha
- orientácia

Spojenia:

- identifikátor
- konce spojenia
- váha

4.4 Špecifikácia požiadaviek na simuláciu

Na ukážku simulácie pomocou aplikáciou vytvoreného modelu PN sú kladené nasledovné požiadavky:

- plynulá simulácia
- grafický pútavá simulácia
- krokovanie dopredu a dozadu po 1 kroku
- spustenie, zastavenie, pozastavenie a resetovanie simulácie v akomkoľvek čase

5 Návrh

Výstup projektu bude aplikácia navrhnutá tak aby spĺňala, pokiaľ možno, všetky vyššie spomenuté body špecifikácie požiadaviek.

Aplikácia bude rozčlenená do niekoľkých menných priestorov (namespaces). Tento spôsob je nanajvýš výhodný pre prácu v tíme programátorov, pretože pri oddelenej práci na jednotlivých častiach aplikácie odpadá problém s kolidovaním rovnako pomenovaných premenných, metód, funkcií, štruktúr, enumerátorov a pod. Keďže každá časť má vlastný menný priestor, vytvoríme tzv. potencionálny rozsah platnosti v rámci deklaratívnej oblasti, čím sa nemusíme osobitne dohadovať na pomenovaní týchto deklarácií.

5.1 Návrh používateľského rozhrania

Používateľské rozhranie je navrhnuté tak, aby ponúkalo maximálnu prehľadnosť a intuitívnosť a zároveň sa neznižovala funkcionalita a možnosti aplikácie. Hlavné okno aplikácie bude rozdelené na šesť hlavných častí: hlavné menu aplikácie, panel nástrojov, panel elementov PN, panel vlastností elementu PN, panel kreslenia a panel informačného textového výstupu. Aplikácia bude mať SDI grafické rozhranie a bude v anglickom jazyku.

5.1.1 Hlavné menu aplikácie

Hlavné menu aplikácie sa bude nachádzať v priestore pod hornou lištou okna aplikácie, teda na štandardnom mieste. Bude obsahovať šesť pod-menu a to File, Edit, View, Tools, Settings a Help.

Pod-menu File bude obsahovať:

- New – vytvorenie prázdneho modelu siete
- Open – otváranie uložených modelov sieti
- Save – uloženie aktívneho modelu siete
- Save As – rozšírená funkcia Save
- Close – zatvorenie aktívneho modelu siete

- Page Settings – nastavenie strany pre tlač
- Print Preview – ukážka pred tlačou
- Print – tlač modelu siete
- Pole 5 naposledy otvorených modelov sieti
- Exit – ukončenie práce s aplikáciou

Pod-menu Edit bude obsahovať:

- Back – návrat o krok späť
- Copy – kopírovať do clipboard
- Cut – vystrihnúť do clipboard
- Paste – vložiť do clipboard

Pod-menu View bude obsahovať:

- Toolbars – menu pre zobrazovanie a schovávanie jednotlivých panelov aplikácie
- View Full Screen – zobrazenie aplikácie na celý monitor
- Zoom – zobrazenie v škále

Pod-menu Tools bude obsahovať:

- funkcie prislúchajúce k práve aktívnemu modelu siete ako ovládanie simulácie, výstupov a pod.

Pod-menu Settings bude obsahovať:

- prístup k rôznym nastaveniam a aktualizáciám aplikácie

Pod-menu Help bude obsahovať:

- Help – manuál k aplikácií
- About – okno s informáciami o aplikácií

5.1.2 Panel nástrojov

Panel obsahujúci nástroje pre práve aktívny model siete, bude obsahovať napríklad ovládacie prvky simulácie, niektoré funkcie hlavného menu (pre rýchlejší prístup), ovládanie zoomu, ovládacie prvky výstupov. Jeho poloha bude priamo pod hlavným panelom.

5.1.3 Panel vlastností elementu

Panel zobrazujúci vlastnosti práve aktívneho elementu PN siete. V tomto panely bude možné tieto vlastnosti aj modifikovať, nie však počas behu simulácie. Bude sa nachádzať v pravej dolnej časti okna aplikácie, pod panelom elementov PN siete.

5.1.4 Kresliaci panel

Okno, kde si používateľ bude môcť pomocou elementov navrhnúť a nakresliť vlastný model PN siete systémom Drag and Drop s pomocou panelu elementov PN. Element bude možné umiestniť kamkoľvek v rámci panelu. Ide o hlavný panel okna aplikácie, ktorý bude za každej situácie viditeľný.

5.1.5 Panel informačného textového výstupu

Panel nachádzajúci sa v spodnej časti okna, má pre používateľa informatívny charakter. Bude viditeľný iba počas behu simulácie alebo spracovania výstupu a bude informovať o ich priebehu a prípadných chybách.

5.2 Formát ukladania modelov siete

Navrhneme XML štruktúru, ktorá umožní užívateľovi vytvoriť ľubovoľný tvar Petriho siete, ktorá bude spĺňať nami definované možnosti a pravidlá. Program tento súbor prečíta, overí správnosť vstupu a vygeneruje z neho funkčnú PS ktorá sa bude simulovať.

5.3 Simulácia

Po preverení, že sieť spĺňa všetky zadané podmienky, bude možné sieť simulovať. Simulácia bude prebiehať ako graficky, tak sa bude priebeh ukladať do štruktúrovaného súboru. Užívateľ bude teda môcť sledovať priebeh simulácie ako vizuálne, tak bude môcť pomocou súboru sledovať históriu celej simulácie.

5.4 Analýza Algoritmov

5.4.1 VRP – riadenie vozidiel

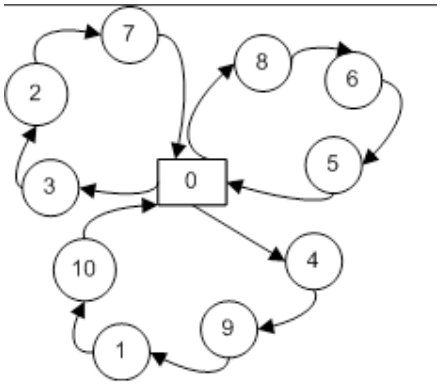
Vehicle Routing problem (VRP) je všeobecný názov priradený triede problémov, ktoré sa zaoberajú priraďovaním ciest vozidlám, ktoré musia z určitého bodu obslúžiť viacerých geograficky roztrúsených zákazníkov pri zachovaní minimálnych nákladov.

VRP problém sa vyskytuje najmä v oblastiach, ktoré sa zaoberajú prepravou, distribúciou a logistikou. Využívanie týchto metód v komerčných sférach ukázalo významné zníženie nákladov od 5% do 20% v týchto oblastiach. VRP problém je označovaný ako tzv. „NP-Hard“, a teda ťažko riešiteľný.

V realite sa ukázali obmedzenia, ktoré viedli k definícii rôznych typov, resp. variant tohto problému. Tieto varianty sú:

- Každé vozidlo má obmedzenú kapacitu - (Capacitated - CVRP)
- Každý zákazník musí byť obslúžený v určitom časovom rozmedzí - (VRP with time windows – VRPTW)
- Predajca využíva viaceré depá , resp. sklady pre zásobovanie zákazníkov – (Multiple depots VRP – MDVRP)
- Zákazník môže vrátiť tovar naspäť do skladu – (VRP with Pick-Up and Delivering – VRPPD)
- Zákazník môže byť obslúžený rôznymi vozidlami - (Split Delivery VRP – SDVRP)
- Niektoré hodnoty (čísla zákazníkov, ich požiadavky, čas obsluhy, cestovania) sú náhodné – (Stochastic VRP – SVRP)
- Doručenie tovaru sa môže pohybovať v priebehu viacerých dní (Periodic VRP - PVRP)

Nasledujúci obrázok ilustruje základné VRP, pričom sklad je označený číslom 0 a jednotlivé uzly, príp. zákazníci, ktoré je nutné prejsť sú označené číslami od 1 po 10.



Obrázok 3 - VRP

5.4.2 Problém riadenia vozíkov s Backhaulsom (VRPB – vehicle routing problem with Backhauls)

Je rozšírením klasického riadenia vozíkov (VRP), ktorý zahrňuje obe skupiny zákazníkov, ktorým má byť produkt dodaný a skupinu výrobcov, ktorí potrebujú svoj tovar distribuovať do distribučného centra.

Na riešenie tohto problému sme použili LHBH heuristiku, založenú na GAP (Generalized Assigment problém). Táto metóda predstavuje nový efektívny spôsob a východiskové riešenie pre nájdenie individuálnych ciest. VRPB tiež známy aj ako linehail-backhaul problém.

Linehaulove (delivery) body sú predurčené na prijímanie istého množstva tovaru z jedného distribučného centra (DC).

Backhaulove (pickup) body sú miesta kde sa vysiela kvantita tovaru naspať do DC. Predpoklad je, že všetky doručenia na každej ceste musia byť ukončené, než sa bude môcť vyslať nový vozík. Sila Backhaulsovho systému spočíva v tom, že prázdny vozík vracajúci sa spať, môže byť naložený a cestou spať, dopraví tovar „inej spoločnosti“ a tým ušetrí náklady na prepravu, čas.

5.4.3 VRP s vyzdvihnutím a doručením (Pick-up and Delivery)

Problém riadenia zariadenia s vyzdvihnutím a doručením je VRP v ktorom sa zvažuje možnosť, že zákazník vráti nejaký náklad. Je treba sa zamyslieť nad možnosťou, že zákazník chce vrátiť tovar v množstve, ktoré sa do prepravného zariadenia nezmestí. Z čoho vyplýva zvýšená vzdialenosť a použitie viacej prepravných zariadení.

Všetky požiadavky sa realizujú z centra, skladiska, a všetky požiadavky na vrátenie tovaru by mali byť smerované s tovarom spať do centra. Číže medzi zákazníkmi nie je žiadna výmena. Každé zariadenie musí doručiť svoj náklad pred tým ako príjme nový.
Cieľ:

Minimalizovať množstvo zariadení v obehu a čas dopravy, s obmedzením na kapacitu zariadenia.

Realizácia:

Riešenie je realizovateľné ak celkové množstvo nákladu priradené jednej ceste, neprekračuje kapacitu prepravného zariadenia, ktoré obsluhuje cestu a PZ ma dosť miesta pre vyzdvihnutie nákladu od zákazníka.

Formulácia:

Schopnosť doručenia: Celkové množstvo nákladu obsluhovaného v ceste, nesmie prekročiť kapacitu zariadenia. Zadaná cesta Ri= $\{v_0,v_1,\ldots,v_m+1\}$ a prepravné zariadenie priradené k nej s kapacitou C. Matematicky $C_d(v_k) <= C$ a $C_d(v_{k+1}) > C$ kde $C_d(v_k)$ je celkové množstvo nákladu dopraviteľné všetkým zákazníkom na ceste, ktorá zažína vo v_0 (centrum, skladisko) a končí vo v_k : $C_d(v_k) = \text{suma}_{vk \in P(1,vk)}\ d_i.P(1,v_k)$,
označuje zákazníka navštíveného pozdĺž cesty od centra k v_k vrátane.

Schopnosť vyzdvihnutia: toto obmedzenie zaručuje, že PZ má dostatok kapacity, na vyzdvihnutie nákladu všetkých zákazníkov na ceste. $C_p(v_k) <= C$ a $C_p(v_{k+1}) > C$; kde $C_p(v_k)$ je celkové množstvo nákladu vyzdvihnutého od všetkých zákazníkov pozdĺž cesty.
$C_p(v_k) = \text{suma}_{vi \in P(1,vk)}\ p_i$

Schopnosť vyzdvihnutia: kapacita PZ môže byť narušená v akomkoľvek uzle cesty. Takéto narušenie závisí od poradia zákazníkov. Nech $L(v_k)$ je náklad hneď po opustení zákazníka v_k. Predpokladajme, že PZ opustila centrum s nákladom $L(1) < C$. Náklad PZ

v ktoromkoľvek bode cesty je $L(v_k) = C_p(v_k) + L(1) – C_d(i_k)$ Náklad PZ daný touto rovnosťou, môže prekročiť kapacitu PZ, čo znamená, že cesta sa stane nespoľahlivou, pretože PZ nedokáže obslúžiť ďalšieho zákazníka v_{k+1} na ceste. Cesta je spoľahlivá ak $L(v_k) < C$ a $L(v_{k+1} > C)$.

5.4.4 VRP s časovými oknami (VRP with time windows)

VRPTW rieši podobný problém ako VRP, s tým rozdielom, že je pridané obmedzenie v podobe časovaného okna, kde toto časované okno je asociované so zákazníkom v, ktorý patrí do množiny V, definujúc časový interval $[e_0, l_0]$, v ktorom môže byť zákazník obslúžený.
Interval $[e_0, l_0]$ v mieste je nazývaný plánovací horizont.

Cieľ:

Cieľom je minimalizovať kolónu vozidiel a sumu času spotrebovaného na cestovanie a času, ktorý je potrebný na obsluhu zákazníkov v požadovaných hodinách.

Realizácia:

VRPTW je , vzhľadom na VRP, charakterizovaný nasledujúcimi obmedzeniami:

- Riešenie je považované za nerealizovateľné ak zákazník má byť obslúžený hodinu po jeho časovanom okne, a teda po dobe vyhradenej pre dodanie tovaru.
- Vozidlo prichádzajúce do cieľovej stanice v predstihu predstavuje zvýšenie času vyhradeného na obsluhu zákazníka počas cesty.
- Každá začatá a ukončená cesta musí byť v súlade s časovým oknom priradeným pre daný bod.
- V „mäkkých“ časových oknách, oneskorený servis nespôsobuje ne realizovateľnosť transakcie, ale je penalizované pridaním hodnoty do výslednej funkcie.

Každému zákazníkovi je priradená cesta určená matematickou funkciou, kde jednou z hlavných podmienok je dodržanie začiatku a príchodu vozidla do stanice v danom časovom okne.

5.4.5 Výber algoritmu v rámci riešenia problému VRP

Keďže hore vyššie uvedené algoritmy riešia rôzne formy problémov prepravy, stanovíme si aj my aké problémy budeme riešiť:

- Kapacita PZ
- Rýchlosť PZ
- Veľkosť PZ
- Dĺžka cesty medzi bodmi
- Vyhnutie sa kolízie v prípade spoločnej cesty PZ
- Optimalizované pre rýchlosť dopravy z centra, depotu
- Vybratie najkratšej trasy, vzhľadom na obmedzenia
- Počtom bodov sa môže riešenie stať „NP-Hard Problem“ t.j. je riešený v nedeterministickom polynomiálnom čase

Existuje veľa algoritmov na riešenie problémov VRP. Väčšina z nich sú heuristické a metaheuristické pretože nie je možné garantovať algoritmus na nájdenie optimálnej cesty v rozumnom čase dosiahnutia tohto výpočtu, v dôsledku NP riešiteľnosti problému.

1. Algoritmy ktoré hľadajú každé možné riešenie pokiaľ je dosiahnuteľné jedno najlepšie:

-Branch and bound (do 100 uzlov) (Fisher 1994)
-Branch and cut

2. Heuristické algoritmy uprednostňujú limitovaný prieskum a typicky produkujú kvalitné riešenie v prijateľnom časovom kvante.

- Savings: Clark and Wright (1964)
- Matching Based
- Multi-route Improvement Heuristics

3. Metaheuristické algoritmy, vyvinuté s cieľom uprednostniť hlboký prieskum v oblastiach, od ktorých sa veľa očakáva. Kvalita výsledku produkovaného týmito metódami je oveľa väčšia ako dosahovaná klasickými heuristickými metódami.

- Ant Algorithms
- Constraint Programming
- Deterministic Annealing
- Genetic Algorithms
- Simulated Annealing
- Tabu Search

Analýza VRP algoritmov nám slúži k lepšiemu prehľadu metód smerovania vozidiel. Aj keď tieto algoritmy neriešia konflikty na cestách, pretože sa predpokladá ich implicitné riešenie pomocou ľudského faktora, napr. Vodičom vozidla, ich princípy je možné aplikovať pri simulácií pohybu vozíkov.

Pre náš projekt však musíme nájsť algoritmus, ktorý sa zaoberá bližšie riešením situácií pri AGV. Existuje niekoľko techník, ktoré riešia práve špeciálne tento typ prevádzky. Pri realizácii je vhodné sa zamerať nie len na smerovanie a riešenie konfliktov, ale aj plánovanie, ktoré im predchádza. Plánovanie má za cieľ najmä zohľadniť určité podmienky, ktoré je nutné pri preprave dodržať, ako napríklad prioritizáciu, alebo finálne termíny, do ktorých sa musí daný stav vykonať. Pričom smerovanie sa sústreďuje najmä na riešenie dvoch základných problémov, a to či cesta existuje, a ak existuje, či je dostupná a bezkonfliktná.

Pri plánovaní a smerovaní teda je nutné dávať mimoriadny pozor na hazardy, ktoré môžu nastať. Sú nimi kolízia, zahltenie, tzv. live-lock-y a tzv. dead-lock-y. Práve pre inšpiráciu pri plánovaní a smerovaní nám poslúžia práve už spomínané VRP. Sú tu síce zásadné rozdiely. VRP sú využívané zväčša pri metropolitných riešeniach, ale v našom prípade úloha musí byť riešená v rámci malého priestoru a teda predstavuje väčšie riziko hazardov. Taktiež kapacitný problém nie je pre AGV relevantný. Napriek tomu zásady smerovania sú aplikovateľné.

5.4.6 Algoritmus pre hľadanie všeobecnej cesty

Algoritmy v tejto kategórií sa snažia zamerať najmä na hľadanie uskutočniteľných ciest pre AGV. Inými slovami, univerzálne smerovacie riešenia ako napríklad bezkonfliktnú časovo najkratšiu cestu.

Prislúchajúce metódy môžu byť klasifikované do troch kategórii:

- Statické metódy, ktoré uzamknú danú cestu, kým sa na nej nachádza vozidlo
- Metódy založené na časových oknách, ktoré uzamknú segment cesty pre určený časový rozsah danému vozidlu
- Dynamické metódy, kde sa používanie segmentu cesty dynamicky priraďuje danému vozidlu

Statické metódy:

Využívajú sa najmä pri menších sieťach. Pri smerovaní sa môže využiť Djikstrov algoritmus hľadania najkratšej cesty, ktorý sa aplikuje tak, aby popisoval obsadenie ciest v čase vozidlami. Potencionálne konflikty sú teda následne riešené cez prepočítavanie novej cesty pričom je vylúčený segment, kde nastáva hazard.

Pri obojsmerných cestách sa môže využiť algoritmus nazvaný PSP, ktorý tiež hľadá najkratšie cesty, pričom segmenty ciest, ktoré už sú obsadené vozidlom, sú vylúčené z prepočtu novej cesty. Nevýhodou týchto ciest, že daný segment je obsadený po dobu kým vozidlo neskončí svoju cestu z východiskového bodu do cieľového.

Metódy založené na časových oknách:

Táto metóda bola vytvorená pre efektívnejšie využitie ciest. Je reprezentovaná značkovaným grafom, pričom každý segment je tvorený uzlom v grafe G. Dva uzly sú spojené ak sú aj cesty susediace. Porovnávaním značkovaní môžeme dosiahnuť časovo najkratšiu cestu. Nevýhodou je výpočtová náročnosť. V tejto metóde sa na rátanie najkratšej cesty môže využiť Djikstrov algoritmus, pričom pre každý uzol je definovaný zoznam časových okien pre dané vozidlá a zoznam voľných časových okien.

Dynamické metódy:

V dynamických metódach sa neuvažujú cesty z globálneho hľadiska, ale iba pre nasledujúci uzol. Nasledujúci uzol je vybraný z množstva susedov, pričom musí spĺňať vždy aktuálne obmedzenia, výber sa teda opakuje dovtedy kým vozidlo nedosiahne cieľ cesty. Nevýhodou takejto metódy je jej neoptimálnosť.

5.4.7 Optimalizácia

Optimalizácia je jednou z najťažších úloh pri hľadaní ciest pri riadení AGV. Využíva sa niekoľko techník. Jednou z nich je:

Intersection graph method

Táto metóda je založená na už spomínanej technike branch - and – bound. Principiálne sa využíva redukovaná množina uzlov v sieti, pričom iba uzly, na ktorých sa nachádzajú križovatky sú využité na nájdenie optimálneho riešenia. V takomto prípade sa nám zmenší na polovicu počet vetiev a sú teda dosiahnuté lepšie výsledky pri čase prepočítavania.

5.5 Verifikácia

Verifikácia je neoddeliteľnou súčasťou modelovanie systému pre reálne nasadenie. Práve verifikácia nám zaručí správnosť modelu. Náš projekt bude na verifikáciu používať program vyvinutý univerzitami v Uppsale a Aalborgu. Ide o program s názvom UPPAAL.

Verifikácia bude riešená importovaním súboru do tohto programu. Súbor bude jedným z výstupov našej aplikácie a bude mať formát kompatibilný s formátom vstupu programu UPPAAL.

Program UPPAAL budeme využívať iba na verifikáciu modelu.

5.5.1 UPPAAL

Uppaal je nástroj na modelovanie, simuláciu a verifikáciu systémov reálneho času vyvinutý univerzitami v Uppsale a Aalborgu. Aplikácia je dostupná na http://www.uppaal.com/ ,kde je možné nájsť aj manuály a odkazy na články publikovane v súvislosti s Uppaal. Systém ponuka konzolovú aplikáciu pre verifikáciu modelov (verifyta) aj grafické užívateľské prostredie, v ktorom je možné grafickou formou modelovať systém, vykonať jeho simuláciu aj spúšťať verifikáciu požiadaviek. GUI je vytvorené v jazyku Java a je spustiteľné pod OS Windows, Linux, Sun , Mac.

Model systému v UPPAALE sa skladá z jedného či viacerých procesov. Popis procesu vychádza z časového automatu , ale umožňuje ho rozšíriť o ďalšie prvky, popis celého modelu je teda rozšírením siete časových automatov.

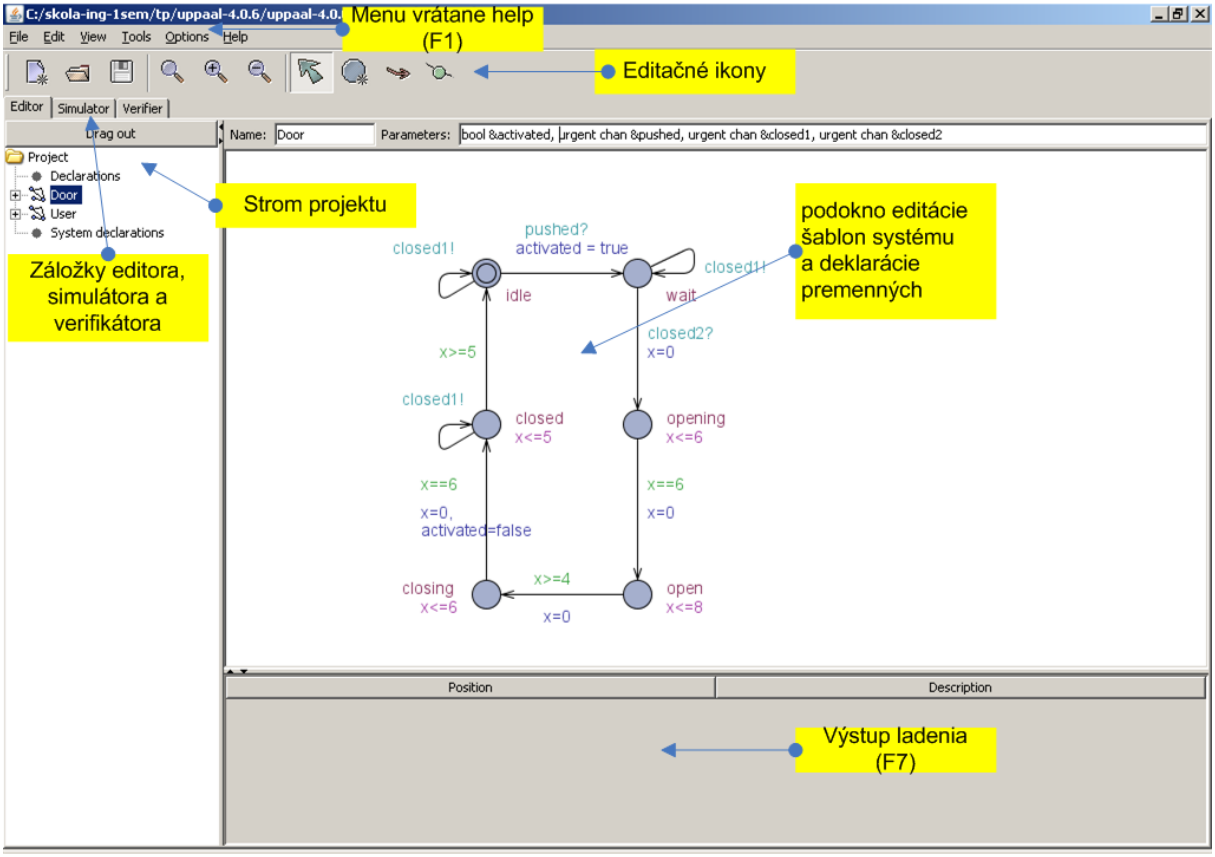
UPPAAL GUI

Sa skladá z 3 časti:

1. Editor systému
2. Simulátor systému
3. Verifikátor systému

5.5.1.1 Editor systému

Editor systému je možné využiť na vytváranie a editáciu systému, ktorý bude analyzovaný. Popis systému je definovaný množinou šablón procesov, globálnych premenných, priradením procesov a definíciou systému(modelu).



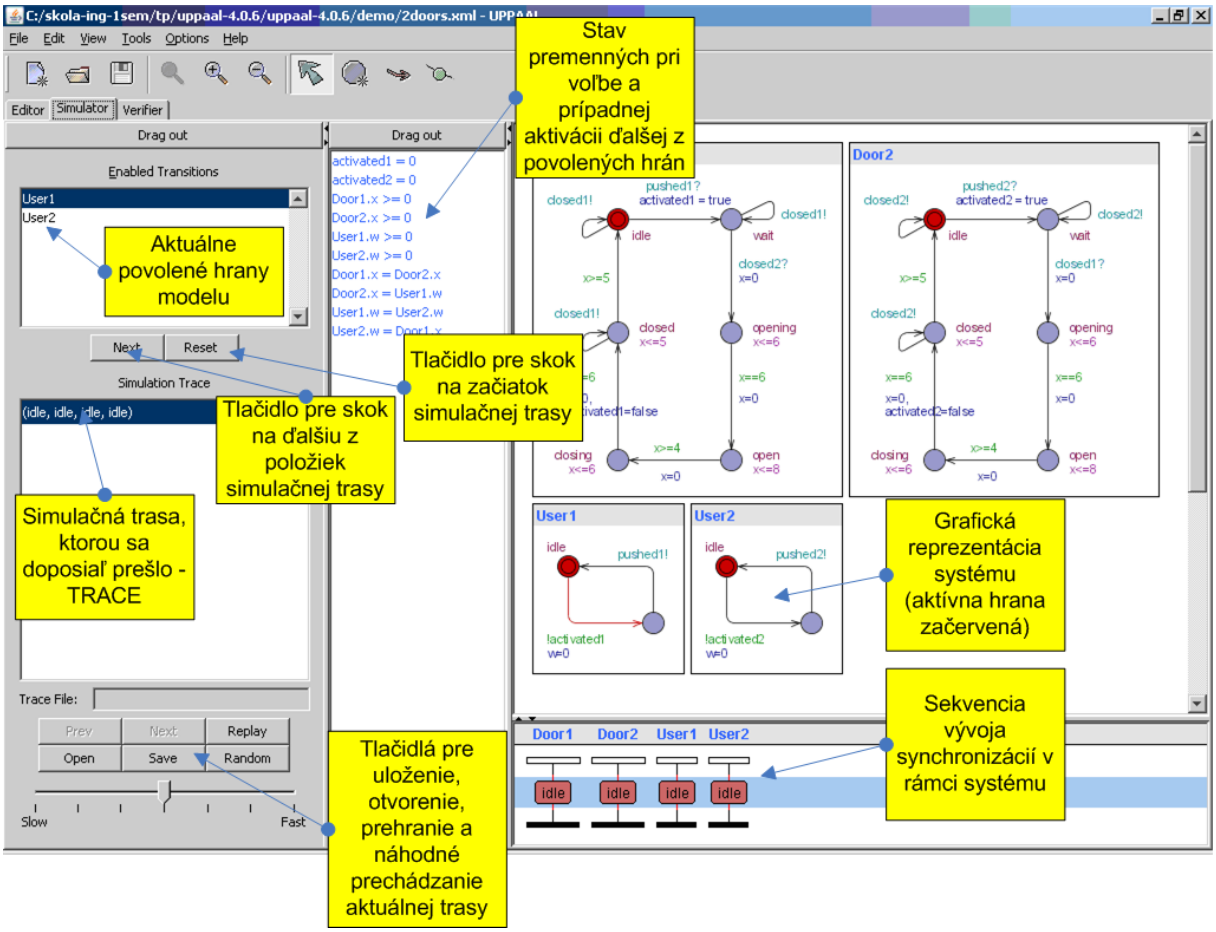
Obrázok 4 – Editor UPAALL

5.5.1.2 UPPAL Simulátor

Simulátor navrhnutého systému automatu umožňuje:

- krokovanie v automate
- sledovaniu vývoja stavu podľa požiadaviek užívateľa
- sledovaniu stavu premenných
- sledovaniu prípadných synchronizácií

Pred vstupom je vhodné skontrolovať syntax − kláves <F7>



Obrázok 5 - UPAALL Simulátor

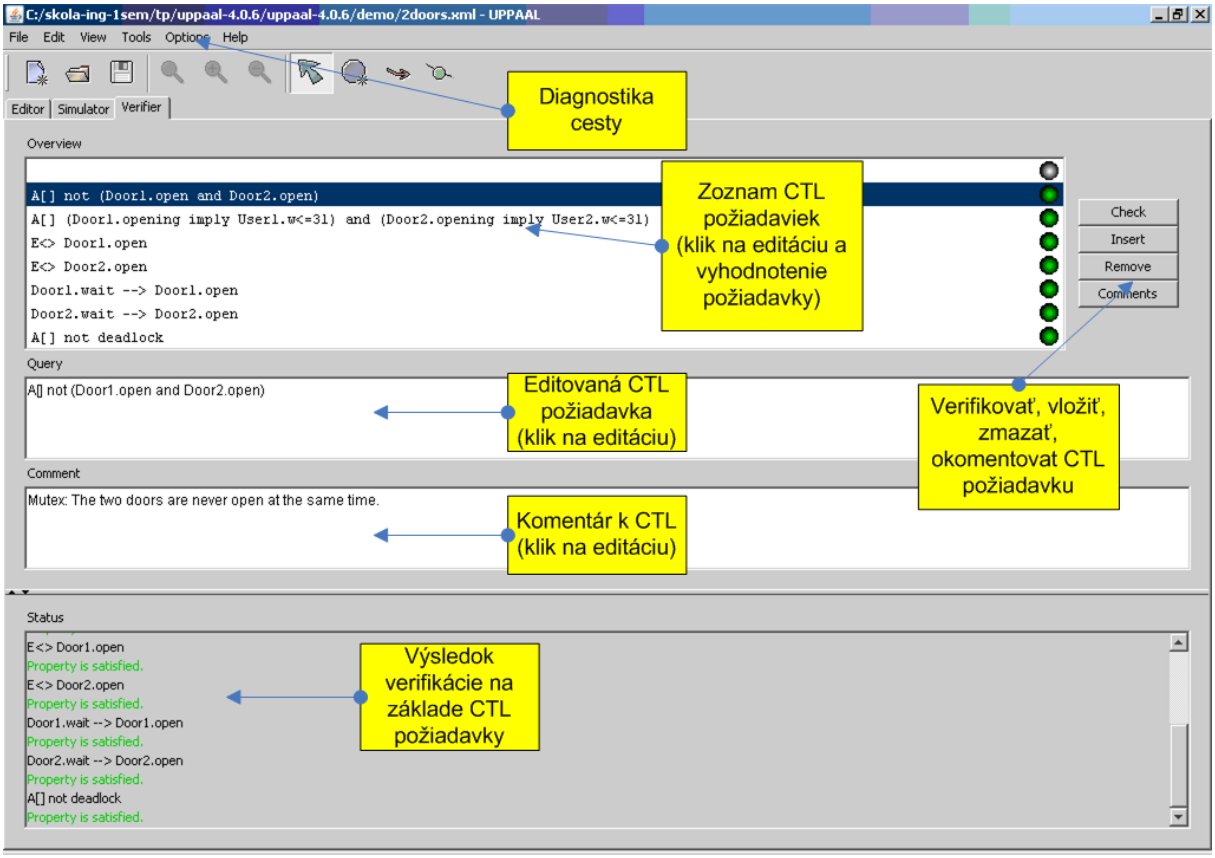
5.5.1.3 UPPAAL Verifikátor

Verifikácia − proces overovania vlastností systému (modelu) vzhľadom ku kritériu s použitím formálnej analýzy. Formálne analýzy sú napr. LTL – linear temporal logic, CTL – Computation Tree Logic.

Na čo slúži?

- Efektívna a formálne správna kontrola správania systému podľa špecifikácie ešte pred realizáciou.
- Správnosť návrhu algoritmov (napr. v telekomunikačnej technike), priemyslové komunikačných protokolov , real-time aplikácií.

Umožňuje kontrolu invariantov a vlastnosti akou je živosť počas skúmania stavového priestoru systému v rámci symbolických stavov predstavovaných obmedzeniami.



Obrázok 6 - Verifikátor systému

Uppaal obsahuje nástroj pre verifikáciu modelu pomocou CTL. Slúži k verifikácii modelu pomocou temporálnej logiky.

príklad A[] not deadlock

− pre všetky miesta platí že v nich nie je deadlock (nie je možné ísť ďalej)

E<> P1. Neaktívny

− existuje stav v ktorom sa systém dostane do miesta P1.Neaktívny

výstupom je TRUE/FALSE

− k nájdeniu cesty k miestu v ktorom je podmienka splnená

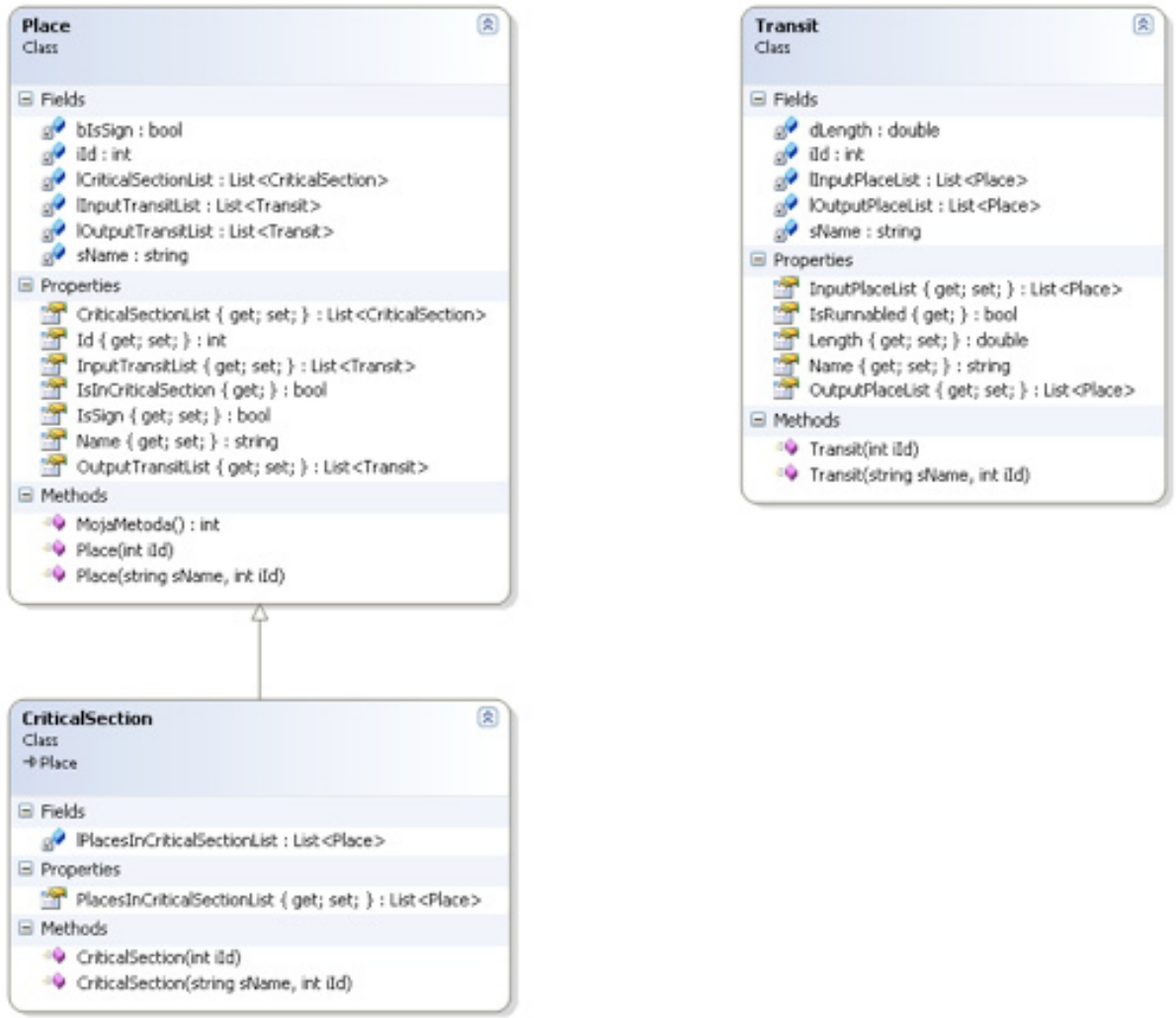
V akom stave došlo ku splneniu alebo nesplneniu podmienky je možné spätne zobraziť v Simulátore.

5.6 Prototyp

Prototyp našej aplikácie je rozdelený na dve časti. Jedna časť sa venuje návrhu tried (dátový model) a druhá je GUI aplikácie.

5.6.1 Dátový model

Navrhli sme predbežný dátový model pre našu aplikáciu. Základom sú tri triedy: Place, CriticalSection a Transit. Nasledujúci obrázok bol vygenerovaný pomocou programu Visual Studio 2008 z tried, ktoré v ňom boli naimplementované.



Obrázok 7 - Dátový model

Z obrázka je zrejmé, že trieda CriticalSection je odvodená od triedy Place. Každý objekt z týchto troch tried má unikátny identifikátor Id. To zaručuje, že každý objekt vieme jednoznačne identifikovať pomocou tohto čísla.

Následne si popíšeme význam a použitie jednotlivých tried.

5.6.1.1 Place

Trieda určená pre ukladanie informácii ohľadom miest v Petriho sieti. Každé miesto bude reprezentovať jeden objekt. Dátová štruktúra je navrhnutá tak, aby každé miesto mohlo mať niekoľko vstupných prechodov a niekoľko výstupných prechodov. Tak isto si každé miesto pamätá, či obsahuje značku.

5.6.1.2 CriticalSection

Môže sa stať, že budeme požadovať, aby na niektorých miestach mohla byť iba jedna značka. Typicky napríklad na rázcestí koľajníc. V príslušnom mieste môže byť iba jeden vlak. Teda označíme zvolené miesta v sieti a vložíme ich do jednej kritickej sekcie a program zaručí, že v kritickej sekcii môže byť najviac jedna značka. . Objekt triedy CriticalSection reprezentuje práve jednu kritickú sekciu.

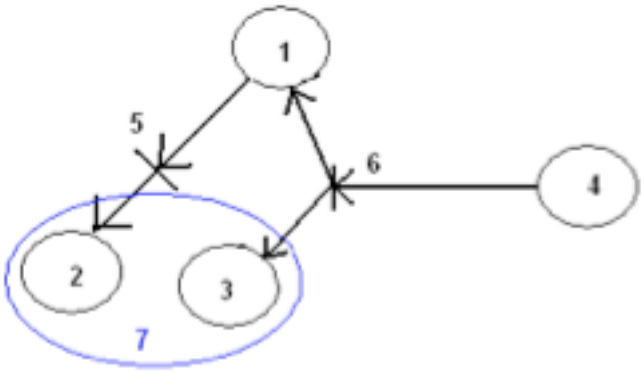
5.6.1.3 Transit

Transit predstavuje prechod medzi miestami. Dátová štruktúra je navrhnutá tak, aby každý prechod mohol mať niekoľko vstupných miest a niekoľko výstupných miest. Každý priechod vie sám zistiť či je spustiteľný, na základe informácii o stavoch jeho vstupných a výstupných miest.

5.6.2 Prepojenie medzi objektmi

Objekty spomenutých tried sú navzájom poprepájané tak, že tvoria celú Petriho sieť a kedykoľvek sa bude počas simulácie dať z nich vygenerovať celú sieť.

Predstavme si situáciu na nasledujúcom obrázku:



Obrázok 8 - Ilustratívny obrázok

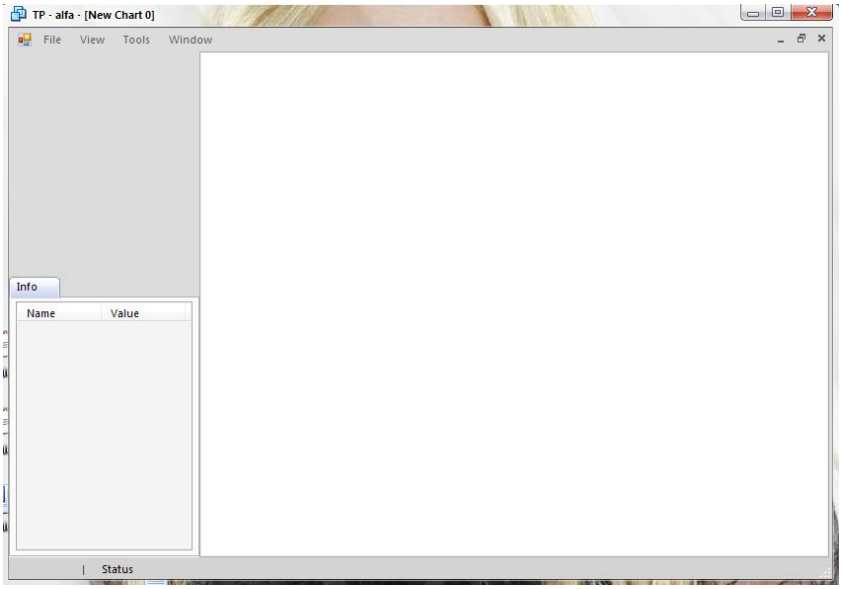
Na obrázku je znázornená časť siete, ktorá sa skladá z 4 miest, 2 prechodmi a jednou kritickou sekciou. Každá z vymenovaných položiek dostala číselný identifikátor. Začneme napr. od objektu s Id rovnej 1. Teda vytvoríme objekt triedy Place, priradíme mu Id = 1. Následne vytvoríme napríklad objekt triedy Transit s Id = 5. Objektu s Id = 1 pridáme ako výstupný prechod Id = 5 a objektu Id = 5 ako vstupné miesto objekt Id = 1. Následne pridáme objekt triedy Place s Id = 2. Jemu pridáme objekt ID = 5 ako vstupný prechod a objektu ID = 5 pridáme medzi výstupné miesta objekt ID = 2. Takto postupujeme, kým nezapojíme celú sieť. Nakoniec vytvoríme Kritické sekcie. V schéme máme jednu kritickú sekciu. Medzi zoznam miest v kritickej sekcii pridáme objekty s ID = 2 a 3 a medzi vstupné prechody objekty ID = 5 a 6. Následne aj objekty Id = 5 a 6 pridajú kritickú sekciu Id = 7 medzi zoznam kritických sekcií, kde sa nachádzajú, pretože predpokladáme, že jedno miesto sa môže nachádzať naraz vo viacerých kritických sekciách.

5.6.3 Prototyp GUI

V prototype GUI môže používateľ vytvoriť ľubovoľnú Petriho sieť pridávaním miest a prechodov. Má možnosť ich pospájať.

Pridávanie miest a prechodov sa dá pravým kliknutím myši na kreslaciu plochu, po ktorom sa mu ponúkne možnosť vytvoriť miesto alebo prechod. Orientované spájanie prebieha dvojklikom ľavým tlačidlom myši na miesto alebo prechod, čím sa inicializuje kresliaca funkcia. Kliknutím na cieľové miesto alebo prechod vznikne spojenie týchto dvoch objektov. Objekty je možné ľubovoľne presúvať po kresliacej ploche. Je ich možné taktiež odstrániť.

Pre spustenie prototypu je potrebné mať na počítači nainštalovaný .Net Framework 2.0



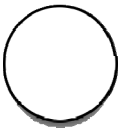
Obrázok 9 - Grafické prostredie po spustení aplikácie

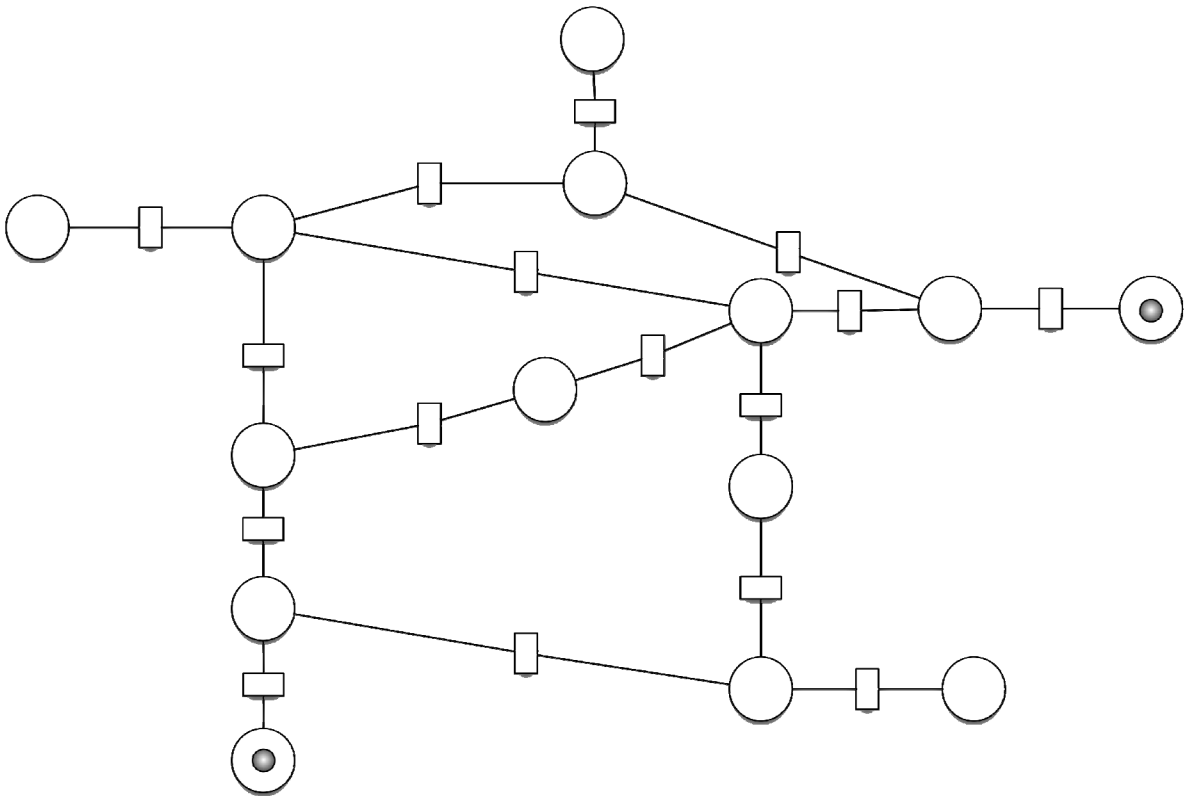
6 Implementácia

V časti implementácia sa v dokumente budeme venovať najmä reálnej implementácií nášho produktu, a teda uvažovaným a použitým algoritmom, štruktúre navrhnutej siete, spojením dátoveho modelu s použitým algoritmom, riešeniam kolízií a konfliktov, možnej optimalizácií a verifikácií. Táto časť dokumentu pokrýva realizáciu výsledného produktu, taktiež popisuje, ktoré časti sme stihli implementovať, ktoré nie a prípadne možné a vhodné vylepšenia.

6.1 Návrh štruktúry siete

Pri návrhu štruktúry siete sme sa rozhodli, že modelovaná sieť nebude modifikovateľná používateľom, ale, že navrhnutý a použitý algoritmus bude program vykonávať na vopred nami stanovenej sieti. Toto riešenie je postačujúce a poskytuje maximálnu možnú a efektívnu prezentáciu použitého algoritmu.

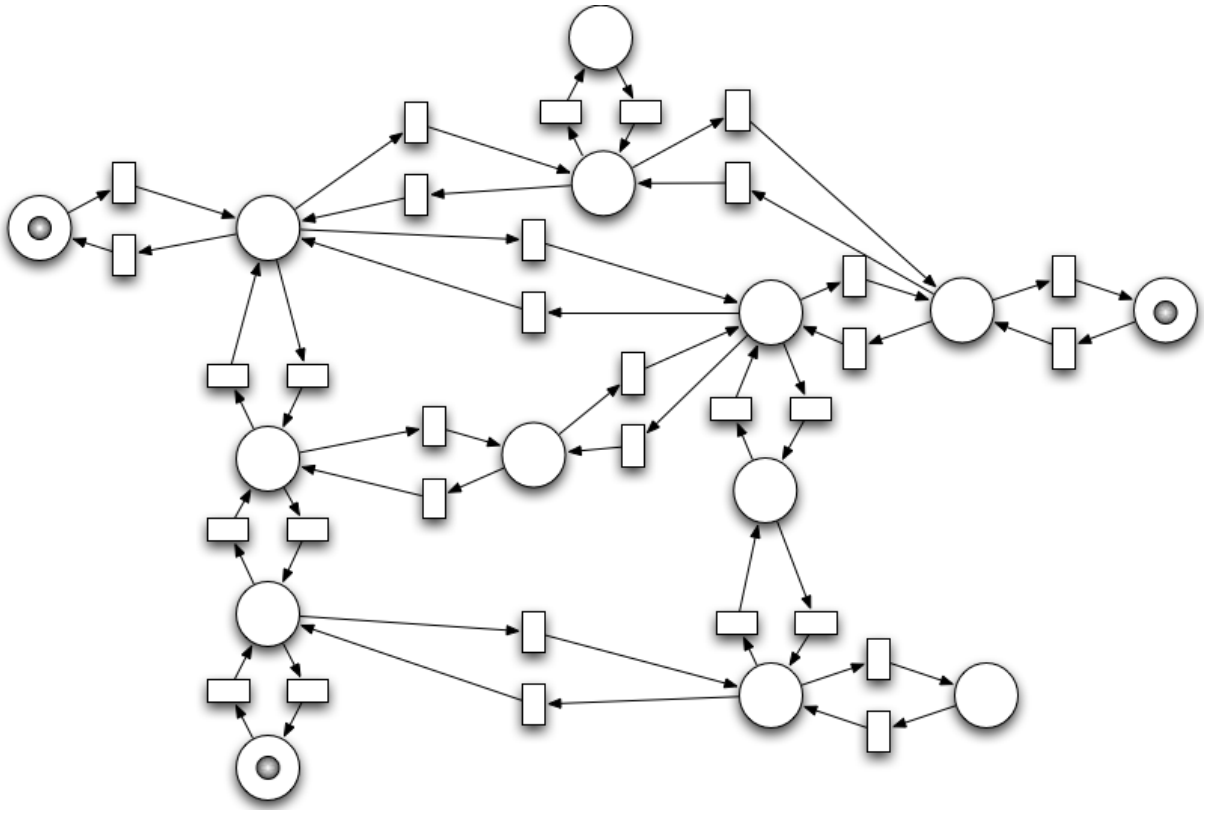
Do implementácie sme navrhli štruktúru siete pre pohyb vozíkov zobrazenú na obrázku č. 10, pričom objekty typu  predstavujú vozíky, objekty typu  predstavujú prechody, objekty typu  predstavujú miesta. Zadefinovali sme, že vozíky sa smú pohybovať oboma smermi, čo je dobre vidieť na modeli Petriho siete v odseku nižšie. Z návrhu teda vypýva, že vozíky sa môžu pohybovať z miesta na iné miesto v oboch smeroch a teda je možný aj návrat na miesto, z ktorého práve vyšiel v prípade, že to bude nutné pre jeho následný pohyb, prípadne aby sa dostal z kolíznej situácie.



Obrázok 10- Základný návrh štruktúry siete

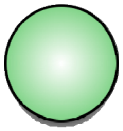
6.2 Petriho sieť

Pre potreby implementácie sme navrhnutú štruktúru siete pretransformovali aj do podoby jednoduchej petriho siete. Na obrázku č.11 môžeme vidieť prepis tejto siete s prechodmi a aj miestami. Je tam taktiež vidieť aj smer akým sa vozíky môžu vydať, ako bolo už vyššie spomenuté, obojsmerne.

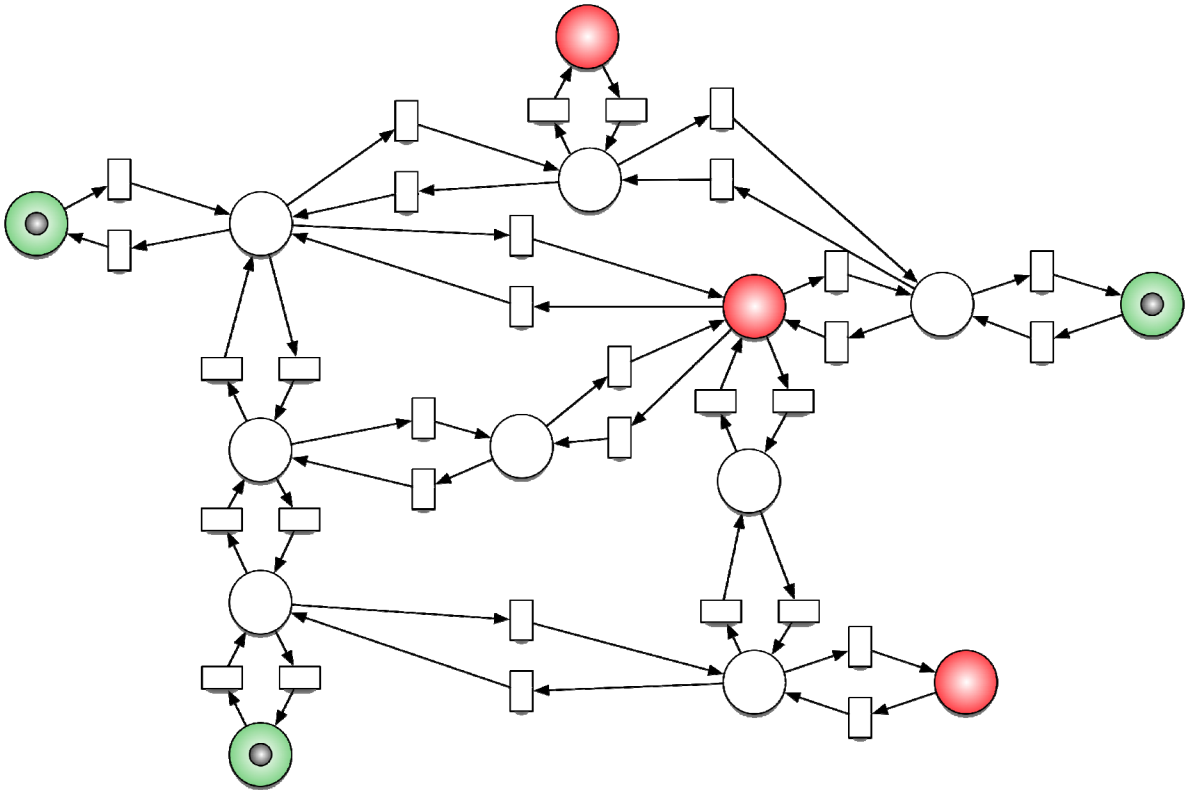


Obrázok 11 - Petriho sieť

Na obrázku č. 12 môžeme vidieť potencionálne počiatočné a konečné miesta, z ktorých vozíky začínajú príp. končia svoju dráhu. Predstavujú teda reálnu štruktúru dráhy podľa, ktorej sa vozíky budú pohybovať, teda jej počiatočné miesto a koncové miesto.

Miesta, zobrazené objektami typu  predstavujú vozíky, objekty typu 

predstavujú počiatočné miesta vozíkov, objekty typu  predstavujú koncové miesta vozíkov pri presune ich dráhou.



Obrázok 12 - Návrh petriho siete s počiatočnými a koncovými miestami

6.3 Použité algoritmy

Algoritmy, ktoré boli použité v implementácií programu je niekoľko. Pri implementácií nášho vlastného smerovacieho algoritmu využívame algoritmus na hľadanie najkratšej cesty. Najznámejší a najoptimálnejsí algoritmus na hľadanie najkratšej cesty podľa ohodnotených hrán je jednoznačne Dijkstrov algoritmus. Smerovací algoritmus vozíkov sme si navrhli sami. Bol však v prevažnej časti postavený na algoritme nazývanom Branch and Bound. Tieto algoritmy sú popísané v nasledujúcich kapitolách.

6.3.1 Dijkstrov algoritmus

Dijkstrov algoritmus je jedným zo základných algoritmov teórie grafov, jeho primárnym využitím je hľadanie najkratšej cesty v hranovo-ohodnotenom digrafe $G = (V, H, c)$.

Pre každý vrchol grafu *G* udržiava algoritmus tri symboly. Prvým je *t(v)*, ktorý zaznamenáva doteraz najkratšiu cestu z počiatku do vrcholu *v*. Druhým je *x(v)*, ktorý si pamätá predposledný vrchol cesty *s – v*, teda vrchol pomocou ktorého sa dostaneme do vrcholu *v* "najrýchlejšie". Tento symbol nie je priamo potrebným pre beh algoritmu, no má svoje využitie pri spätnej konštrukcii cesty z počiatku do cieľa (prípadne hociktorého iného vrcholu množiny *V*). Posledným symbolom je dvojstavová premenná *p(v)* , určujúca či je doteraz najkratšia nájdená cesta *t* konečná alebo ňou nie je. Ďalšou potrebnou charakteristikou bude riadiaci vrchol *r*.

Pred samotným behom je potrebné inicializovať horeuvedené symboly nasledovne:
$t(v) = \infty$, pre všetky $v \neq s$; $t(s) = 0$,
$x(v) = 0$, pre všetky v z V,
$p(v) = false$, pre všetky $v \neq s$; $p(s) = true$,
a za riadiaci vrchol zvolíme počiatok, teda $r = s$.

Samotný algoritmus pozostáva z dvoch opakujúcich sa krokov. Prvý z nich kontroluje či náhodou nie je riadiacim vrcholom vrchol *d*, teda cieľ. V takom prípade algoritmus končí a jeho výsledkom sú *t(d)*, dĺžka najkratšej *s – d* cesty a postupnosť vrcholov *s, ..., x(x(x(d))), x(x(d)), x(d), d*, čo je samotná cesta. Ak však podmienka, že riadiacim vrcholom je *d* neplatí, vykonáme nasledujúci úkon: pre všetky hrany tvaru *(r, i)* z množiny *H*, pre ktoré platí, že $p(i) = false$ a $t(i) > t(r) + c(r, i)$ upravíme symbol *t(i)* na $t(r) + c(r, i)$ a značku *x(i)* nastavíme na *r*.

V druhom kroku nájdeme zo všetkých dočasne označených vrcholov *v* (platí pre ne $p(v) = false$) taký, ktorého *t(v)* je najmenšie. Tento vrchol prehlásime za nový riadiaci a jeho značku *p(v)* nastavíme na *true*, čo bude znamenať, že *t(v)* skutočne označuje najkratšiu cestu z vrcholu *s* do vrcholu *v*. Ak by nastala situácia, že vrcholov s minimálnym *t* je viac, môžeme vybrať ľubovoľný z nich. Ďalej pokračujeme vo výpočte prvým krokom.

Ak na konci výpočtu obsahuje *t(d)* nekonečno (a $x(d) = 0$), tak je zrejmé, že spojenie *s – d* neexistuje.

Vo všetkých behoch prvého kroku sa vykoná maximálne $|H| = m$ operácií, keďže každú hranu použijeme nanajvýš raz. V druhom kroku stačí prezrieť maximálne $|V| = n$ vrcholov. Výpočtová zložitosť Dijkstrovho algoritmu je teda $O(n^2)$.

6.3.2 Branch-and-Bound

Tento algoritmus sa používa na riešenie NP-ťažkých problémov ako napr.

- Knapsack problem
- Integer programming
- Nonlinear programming
- Traveling salesman problem (TSP) Quadratic assignment problem (QAP)
- Maximum satisfiability problem (MAX-SAT)
- Nearest neighbor search (NNS)
- False noise analysis (FNA)

Spolu s Dijkstrovým algoritmom nájdenia najkratšej cesty bude použitý v prípade ak nastane problém kolízie, tento algoritmus sa pokúša nájsť inú cestu od bodu, kde naposledy skončil. Je neefektívne vracať sa tou istou cestou spať, avšak nevyhnutné na odblokovanie vzniknutej možnej kolízie. Efektívnosť prepravy vozíka rieši Branch and bound algoritmus a nájdenie najlepšej cesty zase Dijkstrov algoritmus.

6.3.2.1 Branch (rozvetvenie)

Slúži na rozloženie problému do viacerých dvoch alebo niekoľkých čiastkových úloh., čo predstavuje zjednodušenie pôvodného celkového spôsobu riešenia. Rekurzívnou aplikáciou tzv. krokovania (branch) vzniká stromová štruktúra. Existuje veľa rozličných metód výberu voľby ďalšej voľby bodu, od ktorého sa má zase vykonať rozvetvenie. Vo všeobecnosti sa medzi najznámejšie zaraďujú nasledujúce metódy :

- *Prehľadávanie do hĺbky* – rýchle nájdenie riešenia
- *Prehľadávanie do šírky* – nájdenie kvalitného riešenia, ale nie je také rýchle

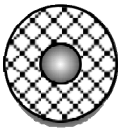
6.3.2.2 Bound (orezanie)

Táto časť má za úlohu, určité vetvy stromu orezať, to znamená v ďalších výpočtoch už sa s nimi nebude počítať, kvôli zrýchleniu výpočtového času.

Tento algoritmus drží v súčasnosti známi publikovaný rekordný čas v počte nájdenia najkratšej cesty. Samozrejme s použitím ďalších algoritmov na nájdenie cesty.

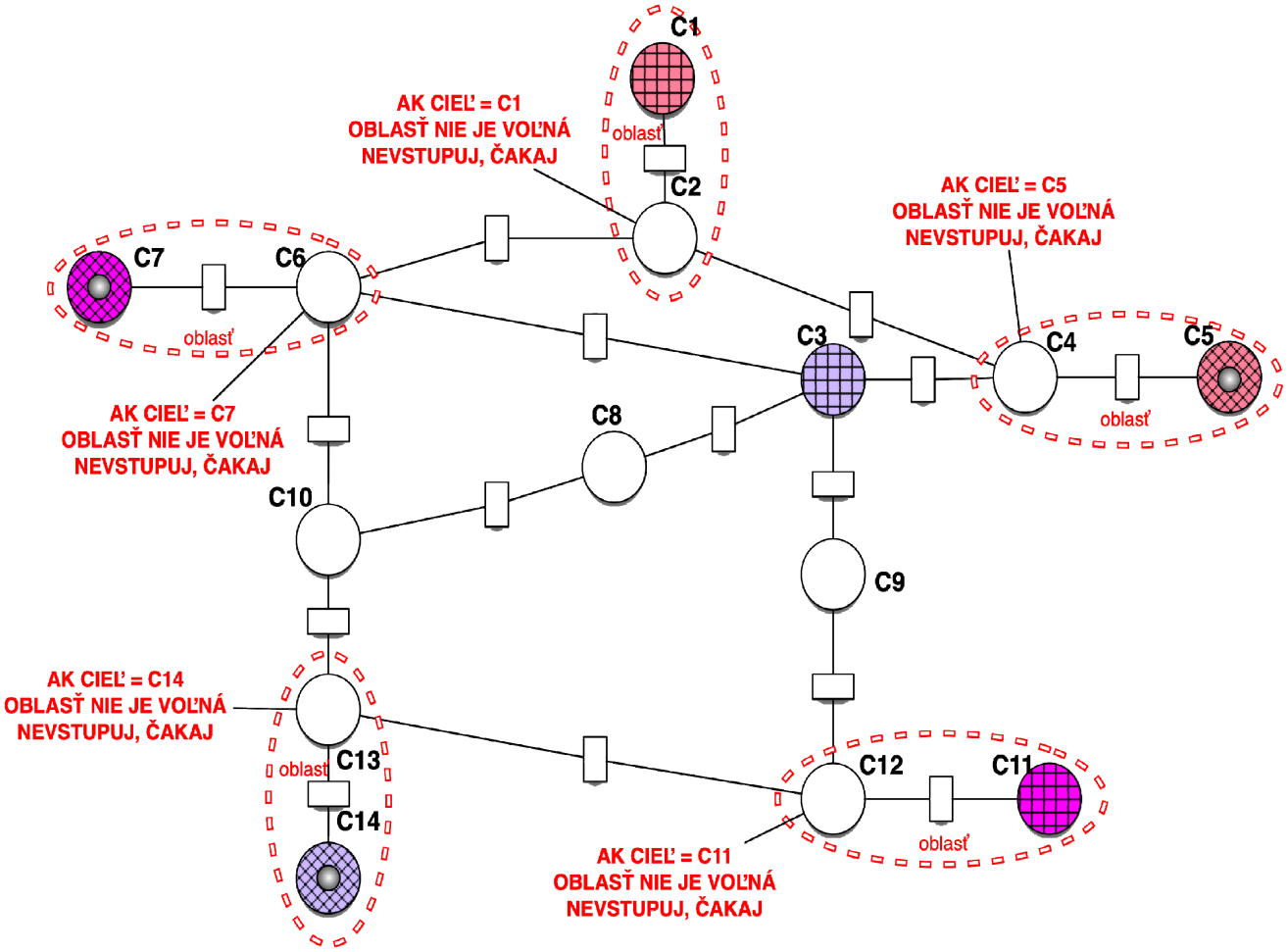
6.3.2.3 Schéma

Schéma znázorňuje zadané štartovacie miesta pre AGV a ich cieľové miesta. Štartovacie body a cieľové body sú kombinované farebne pričom rovnaká farba predstavuje dráhu pre jeden z vozíkov. Vozík sa pohybuje len medzi týmito dvoma bodmi.

Miesta označené nasledovne predstavujú štartovacie miesta  a miesta označené nasledovne predstavujú cieľové miesta pohybu vozíka AGV  .

Na schéme je naznačená simulácia dodávania tovaru na isté skladisko, odkiaľ potom ďalší vozík vyberá iný typ tovaru a zasa ho prepravuje na iné skladisko, alebo vstupný dopravný pás pre spracovanie.

Máme definované vyhradené oblasť, do ktorých môže vstúpiť najviac jeden dopravný vozík, respektíve guľôčka na našej schéme. Dopravné zariadenie smie prejsť cez túto oblasť, len v prípade ak jeho cieľom nie je koncový bod oblasti, v ktorom je už jedno dopravné zariadenie.



Obrázok 13 - Ilustratívna schéma

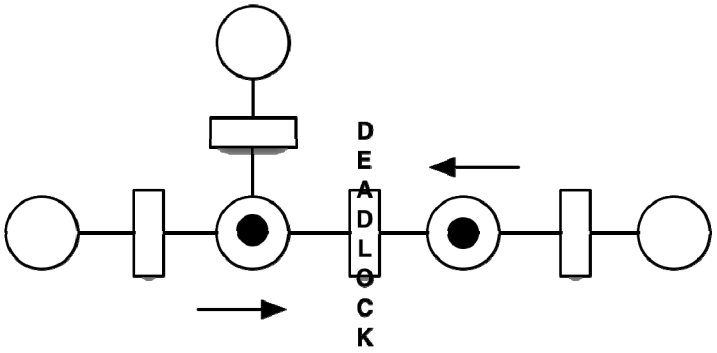
6.3.3 Riešenie Deadlockov

6.3.3.1 Postup riešenia DEADLOCKU algoritmom:

6.3.3.1.1 Vyberie sa PZ s viac ako jednou možnou ďalšou cestou.

BaB zablokuje cestu spať a smer ktorým mal ísť, teda kolízny smer. Dijkstrov algoritmus vyráta novú cestu k cieľu.

Ak cestu nenájde ide náhodne vybratým voľným smerom o jednu pozíciu, dostane úplnu mapu, vyráta sa nová cesta a uspí sa na náhodný časový interval (1-5 sekúnd). Čo by malo zaručiť prejdenie kolízneho PZ sporným bodom a jeho uvoľnenie.



Obrázok 14 - Deadlock 1

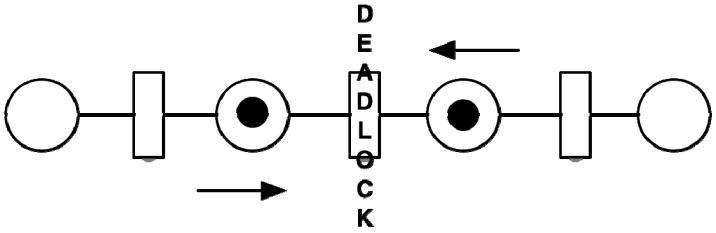
Takýto PZ sa označí špeciálnym identifikátorom „BLACK“ a ak nastane kolízia, ďalšia, vyberá sa práve toto PZ.

6.3.3.1.2 Inak ak je PZ v deadlocku, t.j. nemôže nikam uhnúť, nemá žiadnu inú voľbu cesty z jeho aktuálnej mapy

Vyberie náhodne jedno PZ (prioritu ma „BLACK“) a dostane pre algoritmus celú mapu uzlov a prepojení. Zablokuje sa mu len smer, ktorým chcel pôvodne ísť, kolidujúci smer, toto vykoná BaB a Djikstrom nájdeme novú cestu.

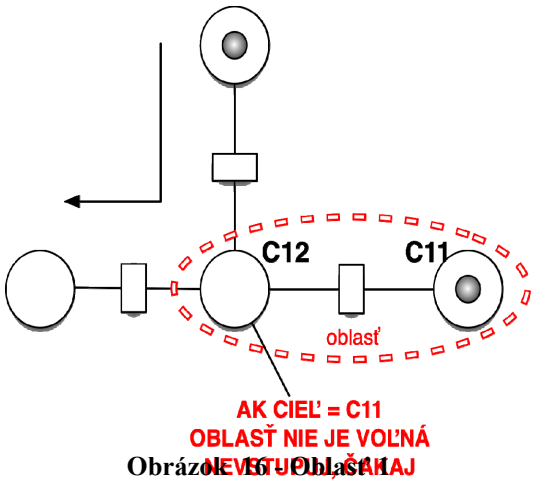
Ak cestu nenájde ide náhodne vybratým voľným smerom (má len jednu možnosť ísť spať) o jednu pozíciu, dostane úplnú mapu, BAB zablokuje cestu kt. tam prišiel, vyráta sa nová cesta k cieľu.

Ak sa cesta nenájde dostane úplnú mapu, vyráta sa nová cesta uspí sa na náhodný časový interval (1-5 sekúnd). Čo by malo zaručiť prejdenie kolízneho PZ sporným bodom a jeho uvoľnenie, alebo opakovane vykonanie kroku 2.



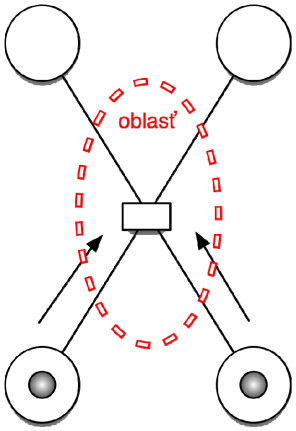
Obrázok 15 - Deadlock 2

Realizácia oblastí:



Obrázok 16 - Oblasť 1

Oblasti sú množiny bodov a prechodov, v ktorých sa môže nachádzať najviac jedno PZ. Špeciálny prípad povoľuje dve PZ ak jedno z nich plánuje len oblasťou prejsť a jedno je už v tej oblasti ale nebráni mu v prechode. Tento typ oblasti nie je nutné definovať, algoritmus si s tým vie poradiť, je to len pre urýchlenie procesu.



Obrázok 17 - Oblasť 2

V tomto prípade sú v oblasti dva prechody. Tieto prechody sú zviazané zámkom, PZ kt. vstúpi prvý na prechod uzamkne ho na čas prechodu, po dorazení do miesta, zámok uvoľní. V našej schéme sa takáto oblasť však nevyskytuje.

6.3.3.2 Postup riešenia deadlocku pomocou supervizora

Jednou z možností riešenia kolízii a konfliktov je využiť supervízorovo riadenie. V našom projekte sme supervízorové riadenie nevyužili, ale poskytujeme ho v dokumente ako alternatívnu možnosť.

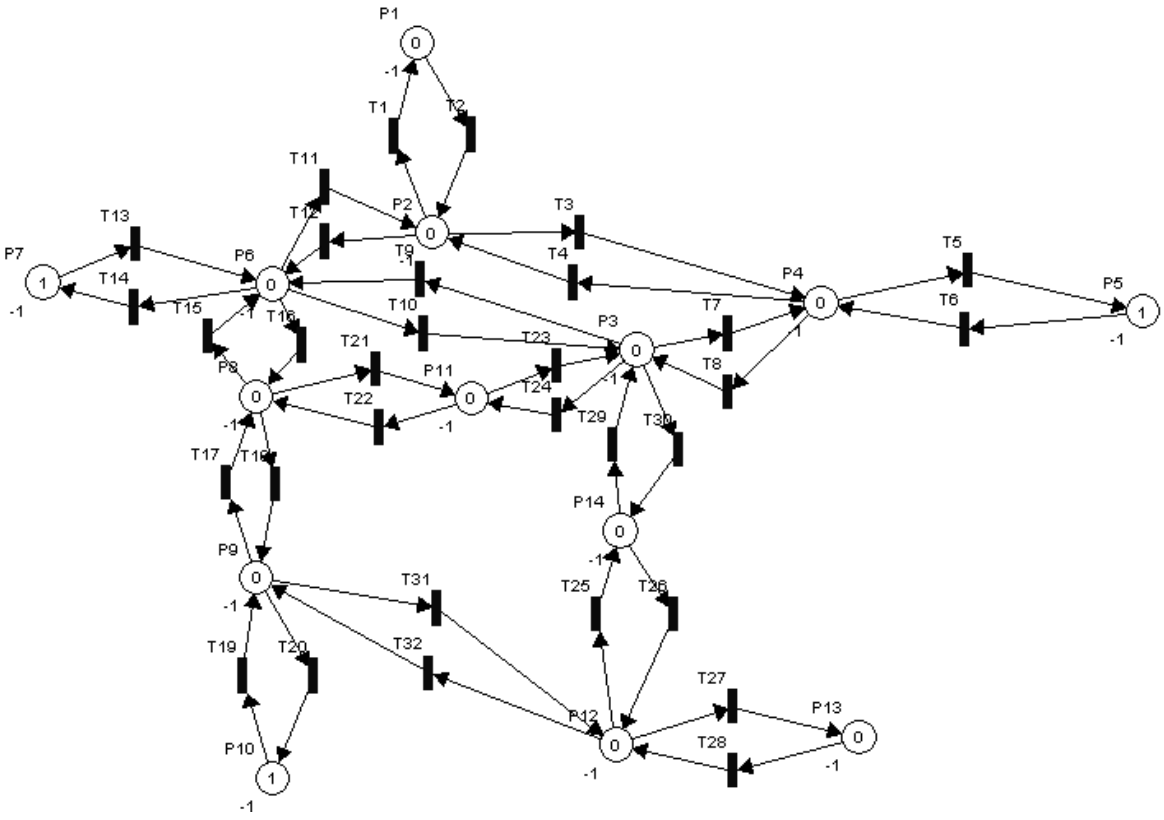
Supervízorové riadenie sa zaoberá automatickou kontrolou diskrétnych udalostných systémov.

Často je nutné regulovať (ovládať) správanie systému, ktorý musí spĺňať bezpečnostné alebo charakteristické kritéria. Napr. zabránenie zrážky automaticky ovládaným vozidlám na poschodí, alebo zabránenie vstupu do niektorých častí týmto vozidlám a pod.

Supervízor zabezpečuje, že správanie modelovaného systému neporuší zadanú skupinu podmienok. Obmedzujúce akcie supervízora sú založené na pozorovaní stavov modelu.

Supervízorové riadenie navrhnuté nad Petriho sieťami spočíva v navrhnutí supervízora, ktorý pozostáva z prídavných miest spojených hranami s prechodmi siete modelujúcej riadený proces. Supervízor zabezpečuje požadované správanie sa modelu.

Náš supervízor bol implementovaný a odskúšaný na nasledujúcom modeli.



Obrázok 18 - Kompletný model

Využili sme metódu založenú na P – invariantoch. Táto využíva vlastnosť Petriho sietí, P – invarianty, na to, aby vypočítala supervízora. Vybrali sme ju z dôvodu, že je

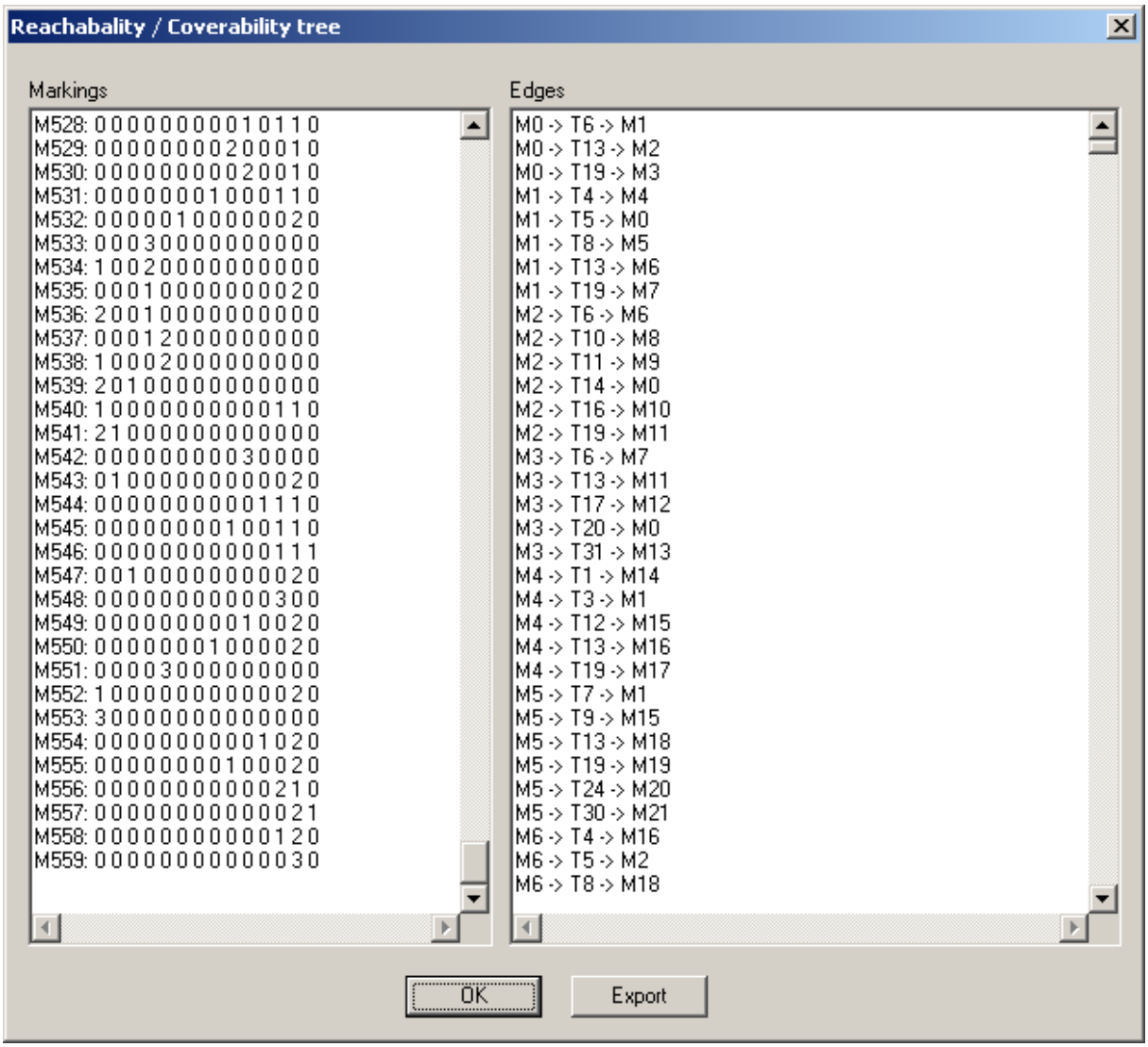
jednoduchá a veľmi efektívna, je len o kúsok náročnejšia ako počítanie s maticami. Z toho dôvodu môže byť supervízor navrhnutý aj pre veľké Petriho siete, otvárajúc možnosti navrhovať supervízory pre veľké a komplexné systémy.

Pre návrh supervízora pomocou tejto metódy bolo potrebné, najprv navrhnúť podmienky
Tieto podmienky kladené na systém museli byť lineárne nerovnosti zložené z prvkov značkovacieho vektora.

$\sum_{i=1}^{n} l_i \mu_i \leq \beta$, kde Kde μ_i je značkovanie miesta p_i a l_i a β sú celé konštanty.

Táto nerovnosť môže byť pretransformovaná do rovnosti pomocou pridania nezápornej premennej μ_c. Táto rovnosť potom vyzerá nasledovne : $\sum_{i=1}^{n} l_i \mu_i + \mu_c = \beta$

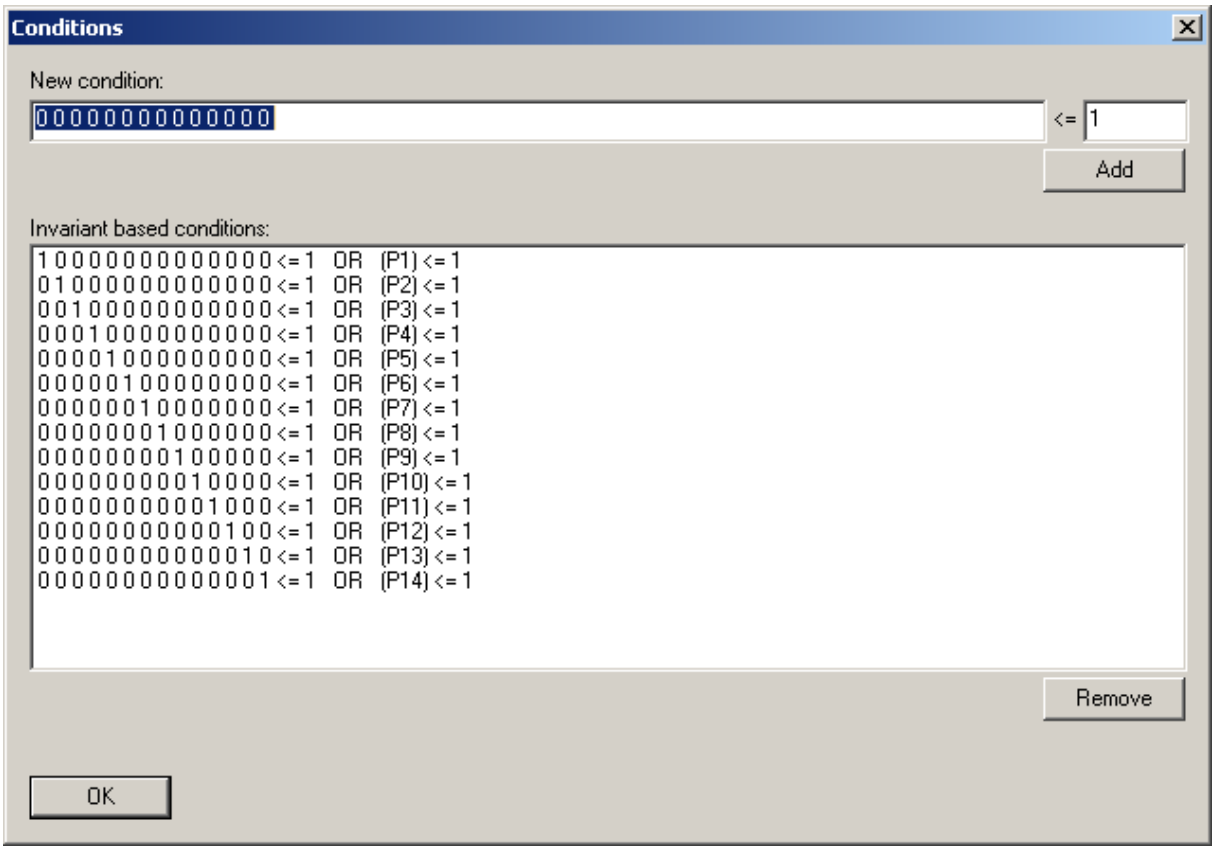
V našom prípade sme na návrh supervízora využili softvér PetriNet Designer. Tento implementuje aj vyššie spomínanú metódu supervízoroveho riadenia založenú na P-invariantoch.



Obrázok 19 - Analýza stromu dosiahnuteľných značkovaní

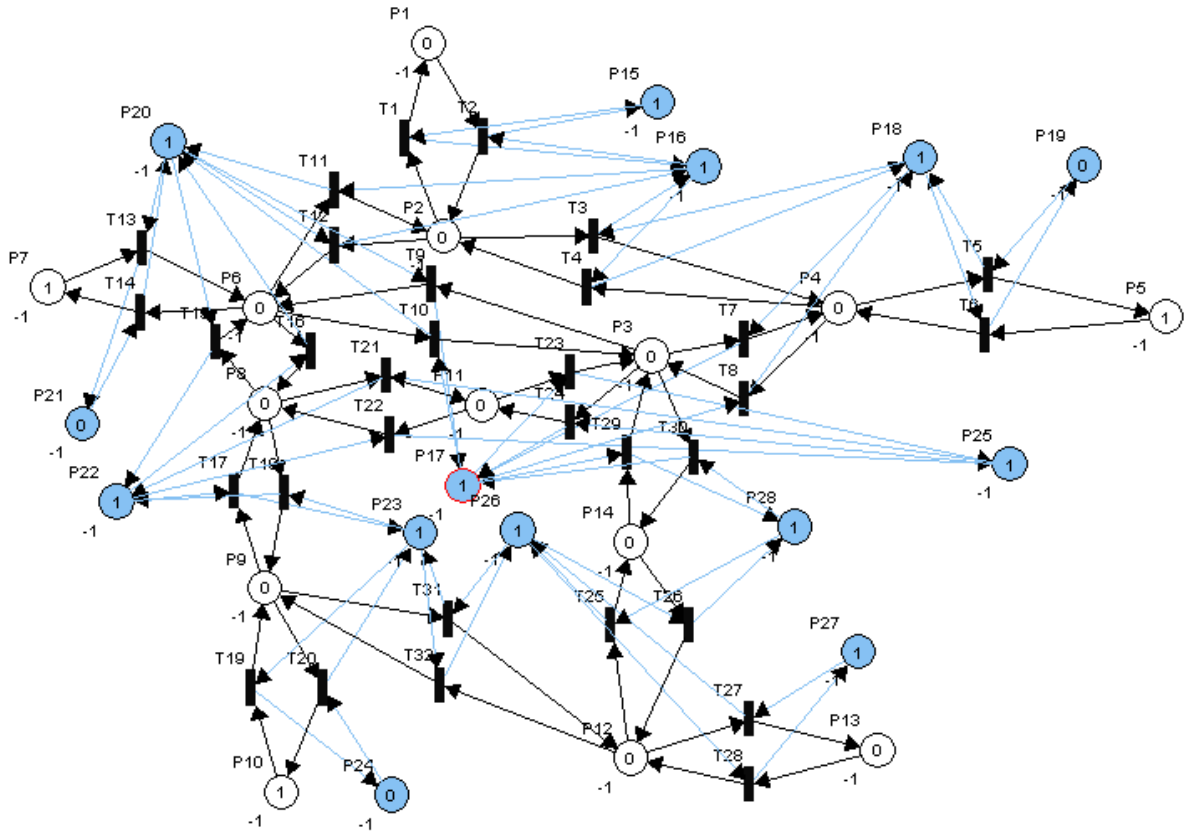
Na obrázku 1. vidno, že v prípade neupraveného modelu bez supervízorového riadenia môže dôjsť ku kolíziám (M559, M558, M557 atd.).

Cieľom teda bolo navrhnúť také supervízorove riadenie , ktoré by zabránilo kolíziám v miestach kríženia. Nebolo však cieľom nahradiť algoritmy na výber najkratšej cesty, pripadne ekonomickosť a efektivitu pohybu vozíkov. Tieto úlohy majú hlavné riadiace algoritmy Dijkstrov a Branch and Bound. V zásade by táto úprava modelu mala fungovať ako dodatočný bezpečnostný prvok(tj. nezávislý zabezpečovací systém) a mala by znížiť riziko kolízie v prípade neočakávaných zlyhaní.



Obrázok 20 - Navrhnuté podmienky pre supervízora

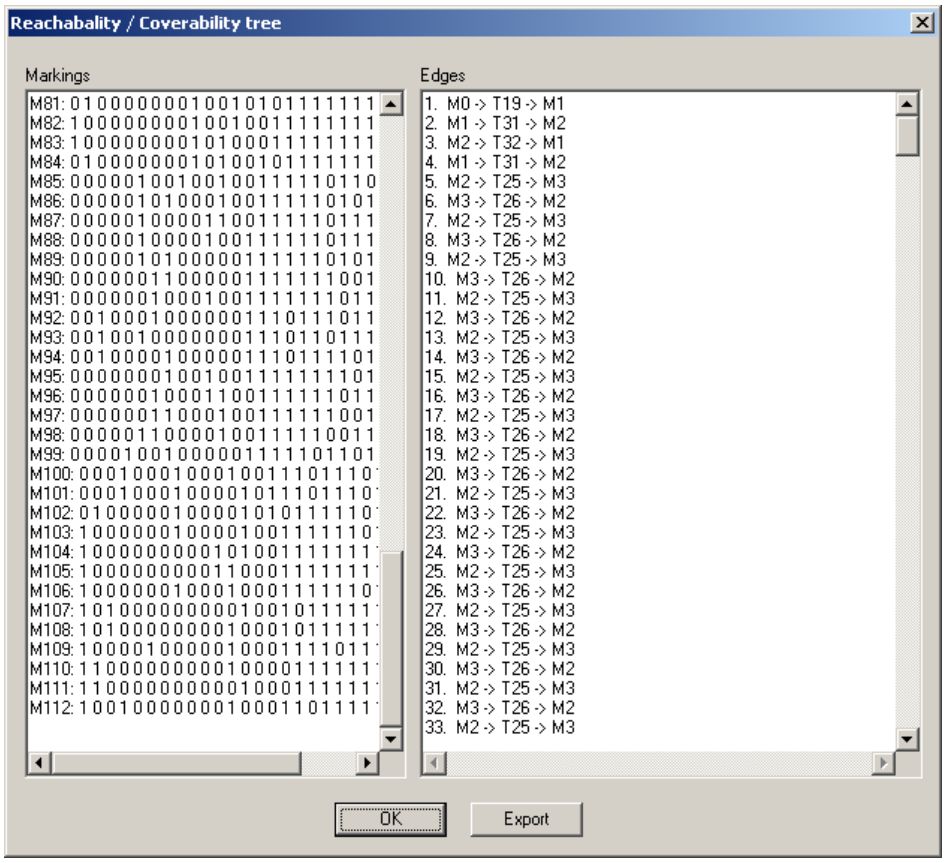
Po zadaní podmienok do programu nám tento vygeneroval novú sieť s implementovaným supervízorom. Pri jej návrhu využil vyššie spomenutú metódu založenú na P-invariantoch.



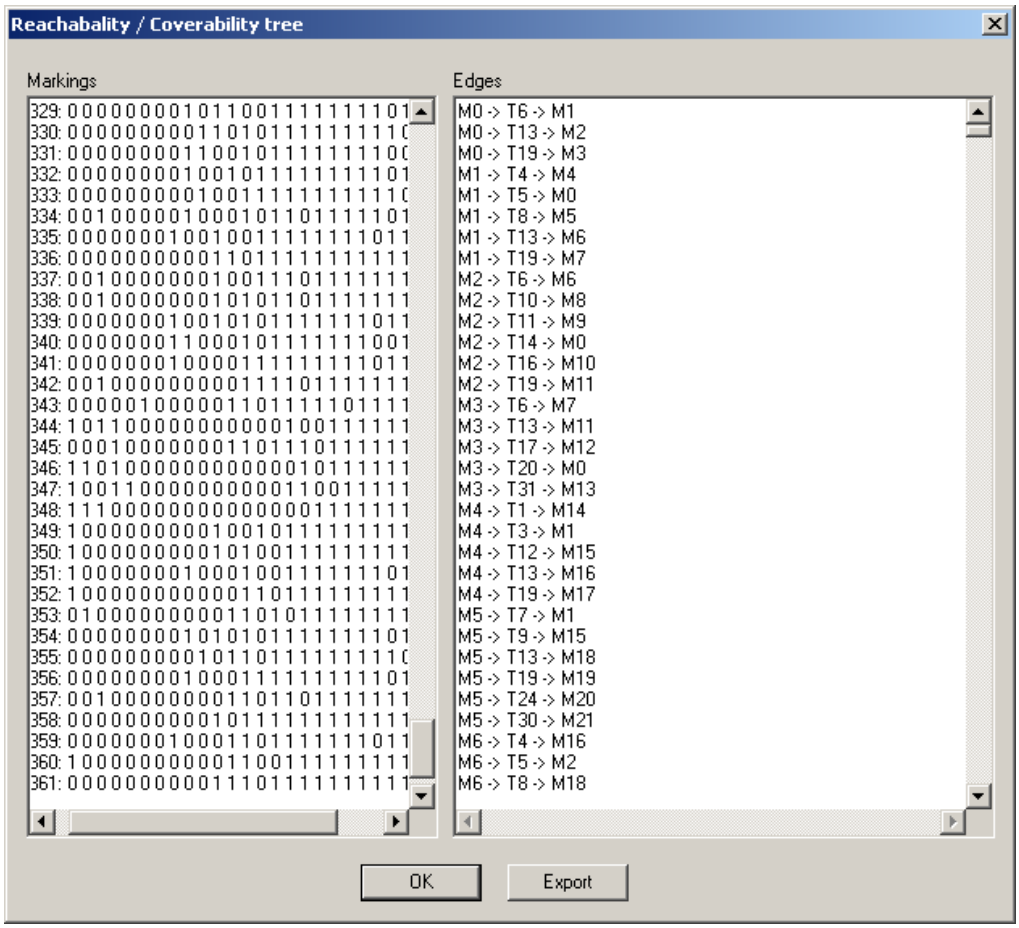
Obrázok 21 - Model s implementovaným supervízorom.

.

Tento sme následne overili pomocou 500 krokovej simulácie spustenej niekoľkokrát za sebou a analýzou stromu dosiahnuteľnosti.



Obrázok 22 - Výsledok simulácie 1



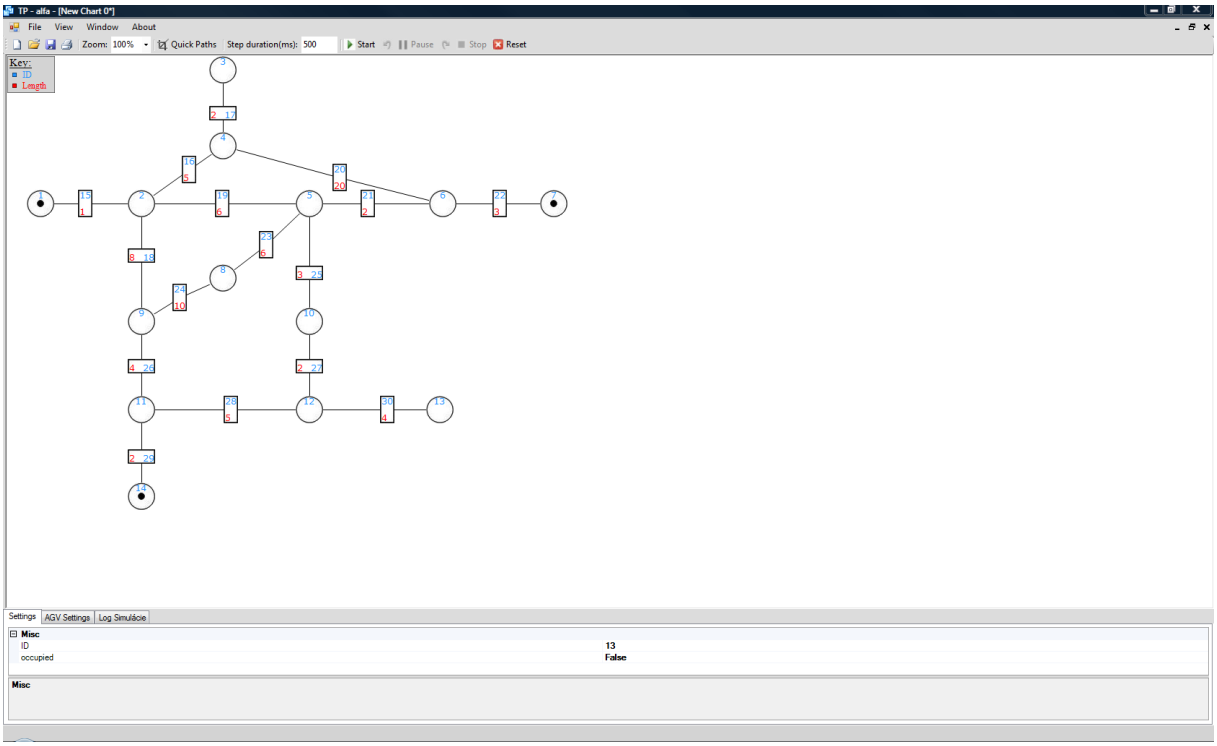
Obrázok 23 - Výsledok analýzy stromu dosiahnuteľných značkovaní

Podľa simulácie by mal byť supervízor správne navrhnutý a funkčný, pričom predstavuje jednu z ďalších dodatočných možnosti riešenia problému kolízii nami navrhnutého modelu.

6.4 Používateľské rozhranie aplikácie

Používateľské rozhranie sa z väčšej časti nezmenilo oproti návrhu a rozhrania prototypu. Napriek tomu i tejto oblasti došlo k niektorým významným zmenám, ktorým sa v tejto kapitole bude venovať.

K prvej zmene z pohľadu štartu aplikácie došlo z dôvodu predvytvorenia zvoleného simulovaného modelu. Zjednoduší sa tak simulácia modelu, aby nebolo nutné jeho vytváranie po každom štarte aplikácie. Druhou zmenou je, že aplikácia sa spúšta maximalizovaná. Pre prehľadnosť grafického rozhrania a simulácie je potrebné rozlíšenie monitora aspoň 1280x1024 pixelov.



Obrázok 24 – grafické rozhranie po štarte

Na obrázku je vidieť všetky dôležité ovládacie prvky aplikácie. Všetky sú viditeľne a rozmiestnené podľa použtia a významu. Zároveň, tým, že sú jasne viditeľné na prvý pohľad, sa zjednodušuje ovládanie aplikácie.

Hlavné menu zostalo nezmenené a ponúka možosti súboru, zobrazenia a okna. Pod ním je menu hlavných ovládacích prvkov. Je rozdelené na tri hlavné časti. Prvou časťou sú základné ovládacie prvky pre prácu so súborom. V druhej časti sa nachídza možosť QuickPaths a Lupa. V poslednej časti sú ovládacie prvky simulácie.

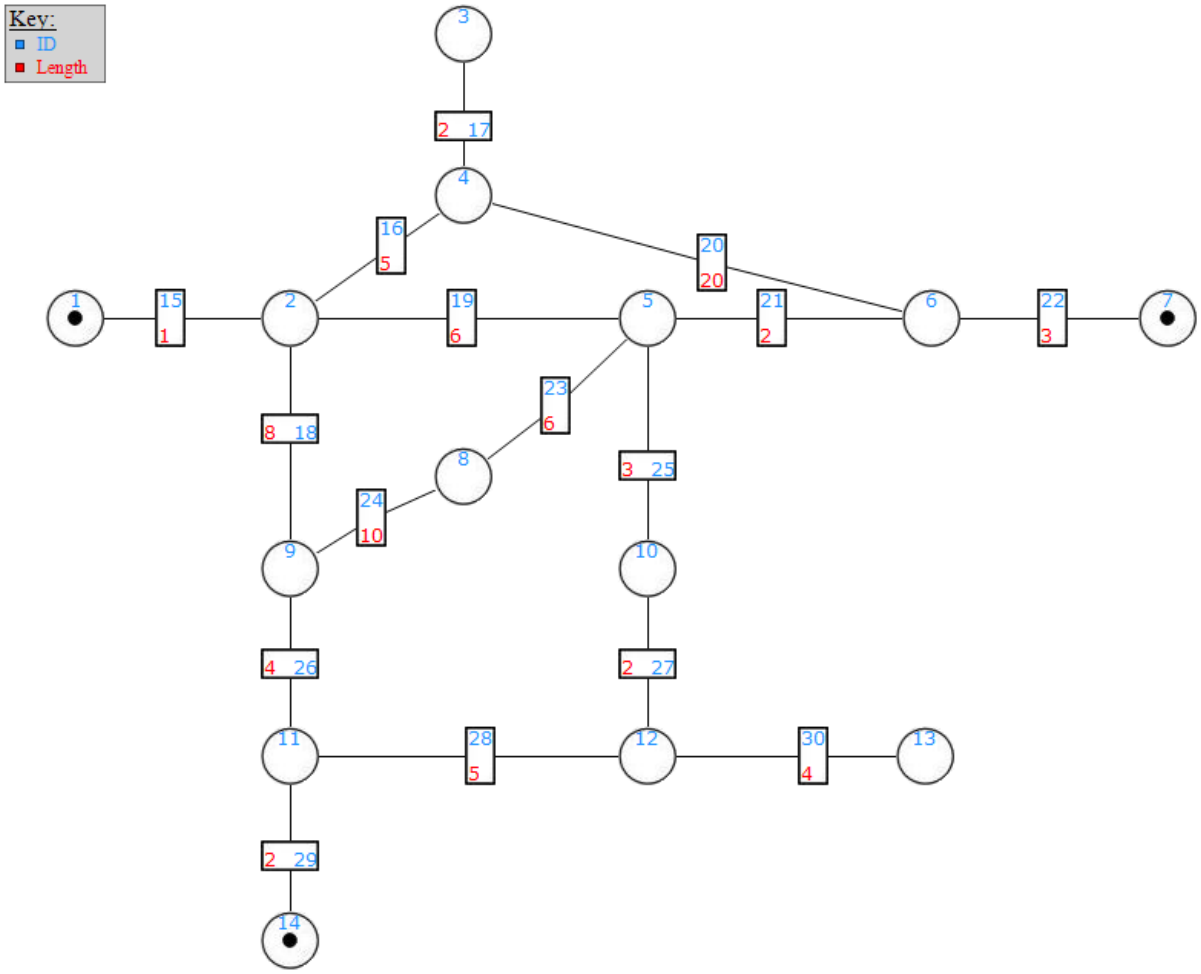
Hlavné okno aplikácie pozostáva z grafického modelu siete. V ňom je možné vytvárať ľubovolnú, užívateľom navrhnutú sieť. Pridávanie miest a prechodov sa dá cez menu vyvolané pravým tlačítkom na bielej ploche náčrtu. Spájanie je možné cez dvojklik na prvok alebo cez menu vyvolané pravým tlačidlom na myši, na konkrétnom prvku. Začne sa vytváranie spojenia, ktoré sa definitívne pridá kliknutím ľavým tlačidlom na cieľový prvok. Odstránenie spojenia nie je momentálne implementované. V ľavom rohu je vidieť legendu.

Na spodku aplikácie sa nachádzajú tri časti. Informácie a editácia vlastností prvku, AGV a log simulácie.

7 Priebeh simulácie zvoleného modelu

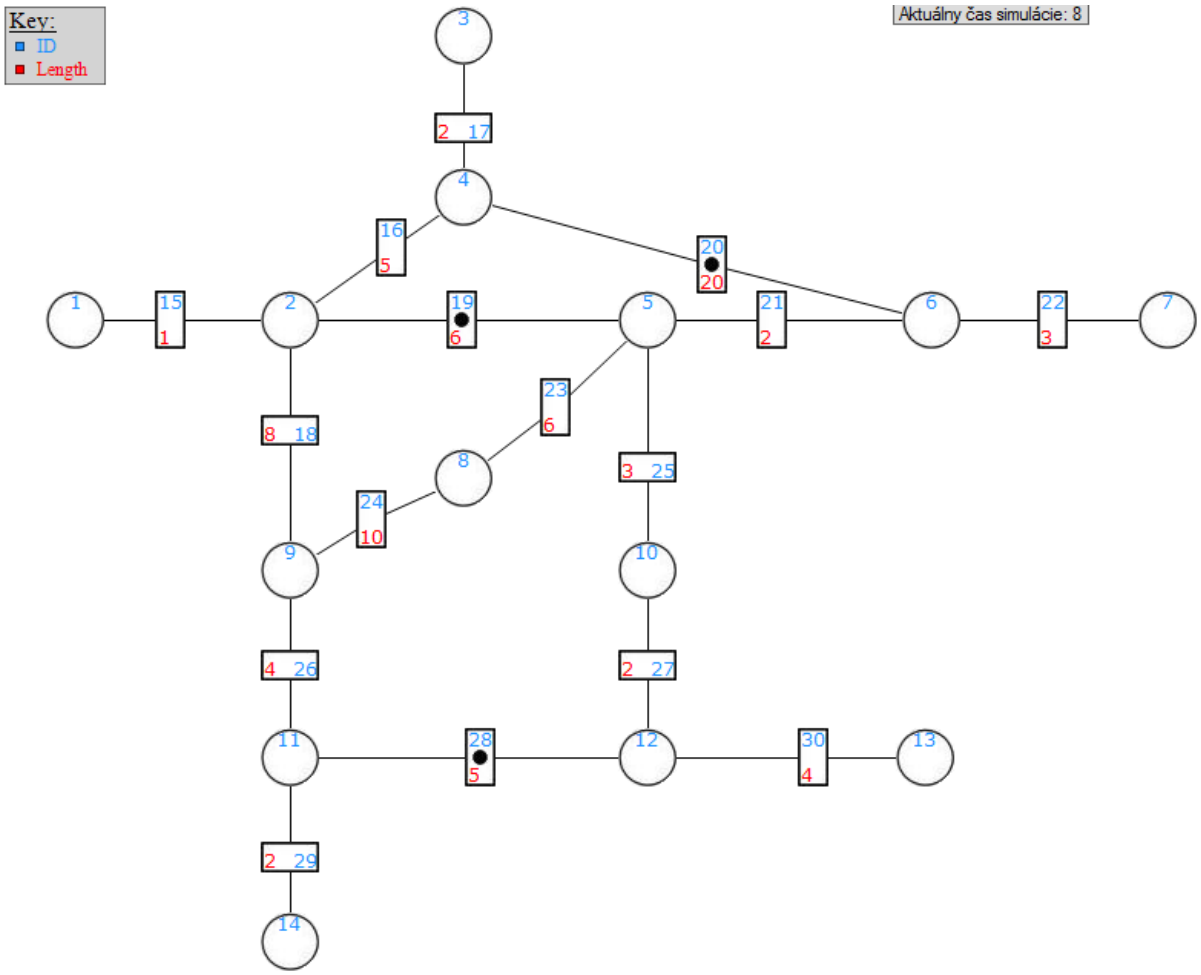
V tejto kapitole bude ukázaný postup simulácie v grafickom rozhraní aplikácie.

Na začiatku simulácie vyzerá model nasledovne.



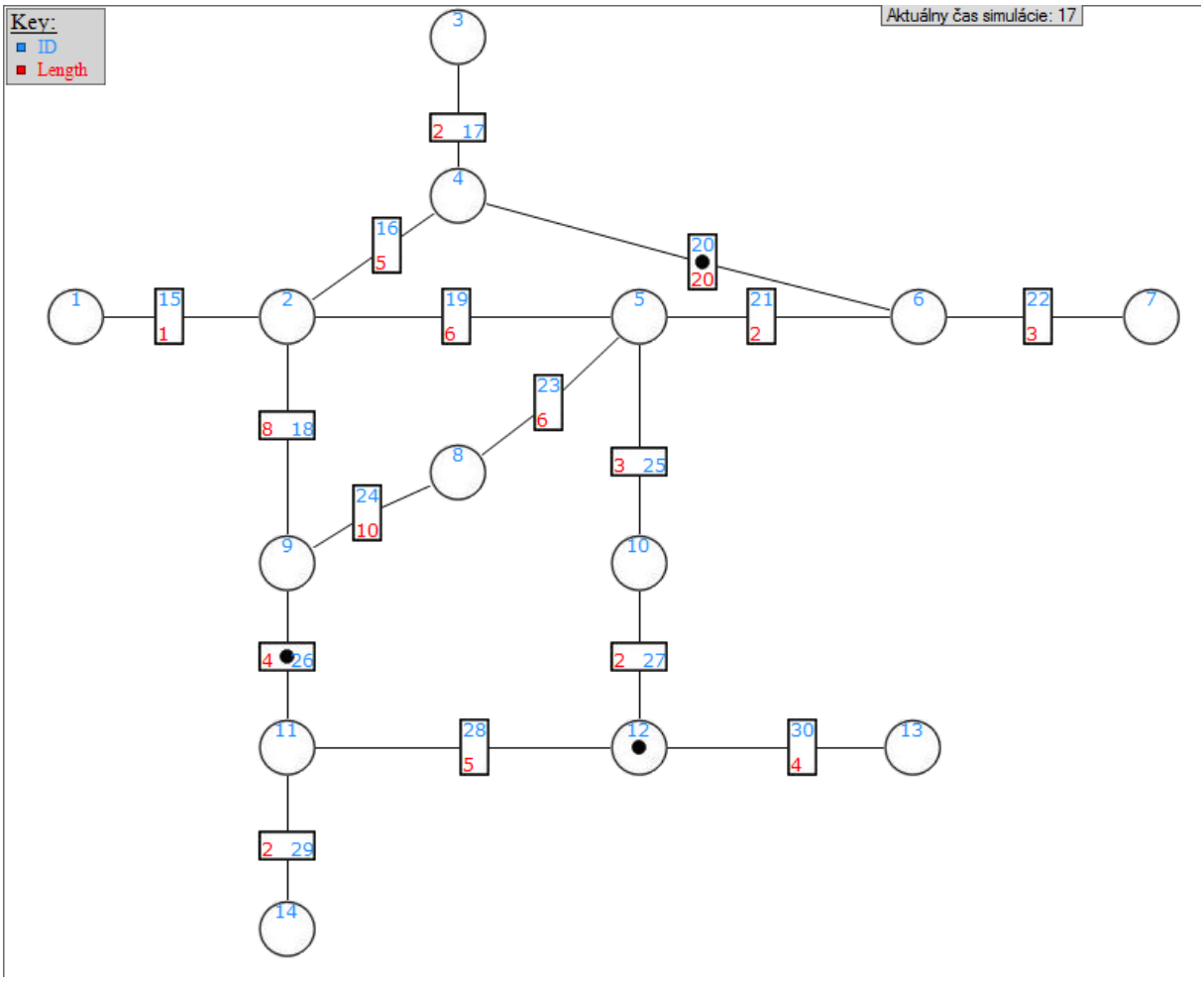
Obrázok 25 – Začiatok simulácie

Na nasledujúcom obrázku je zobrazený niektorý z nasledujúcich krokov simulácie.



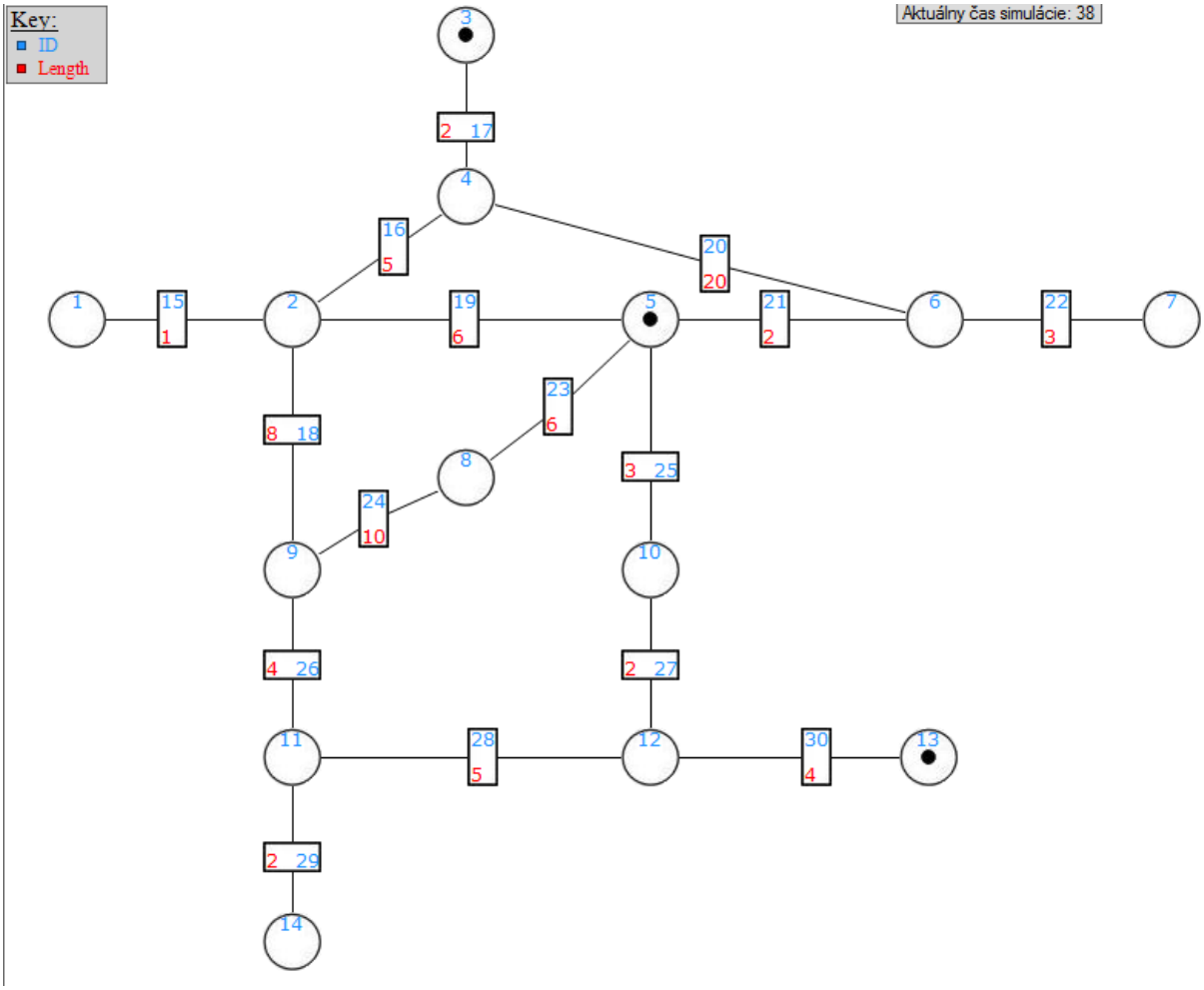
Obrázok 26 – krok simulácie

Pre ukážku vyberem ešte jeden z krokov simulácie.



Obrázok 27 – krok simulácie

Na záver finálny stav simulácie.



Obrázok 28 – Konečný stav simulácie

8 Zhodnotenie

8.1 Zhodnotenie prace počas semestra

Tímovému projektu sme sa venovali dva semestre. V prvom semestri sme sa snažili hloboko ponoriť do témy, ktorá nám bola vtedy tak neznáma, avšak po semestri analýzy a neustáleho štúdia sa nám téma viac a viac približovala. Zanalyzovali sme existujúce riešenia, naštudovali rôzne typy algoritmov, ich efektivitu a možnosť využitia vo výslednom produkte. Navrhli sme prototyp, algoritmy a upresnili sme si celkovú predstavu výsledného produktu.

V nasledujúcom semestri sme sa venovali implementácii, drobným úpravám a testovaniu produktu. Aj keď sme nestihli splniť všetky špecifikované požiadavky na výsledný produkt, je náš výsledný projekt komplexný a plne funkčný v rámci požiadaviek, ktoré sme stihli implementovať. Implementovaný produkt spĺňa hlavné stanované cieľe a je využiteľný. Do budúcnosti nechávame otvorenú otázku pre potencionálnych pokračovateľov ohľadom lepšej možnosti optimalizácie algoritmu, prípadne iných zlepšení. Výsledný produkt je prehľadný a intuitívny na ovládanie.

8.2 Osobný prínos tímového projektu

Riešenie tohoto projektu nám prinieslo veľa nových a zaujímavých poznatkov z oblasí petriho sietí, stavových automatov a ich spracovania pomocou aplikačno špecifických algoritmov. Je vidieť, že táto oblasť má veľké praktické využitie, v ktorej prináša ušetrenie finančných a časových nákladov a v neposledom rade aj automatizáciu procesov. Taktiež každý z nás si určite odniesol aj bohaté skúsenosti z práce v tíme, ktorá je nevyhnutná najmä do praxe v našom budúcom profesionálnom živote. Pochopili sme aké je nevyhnutné pracovať spoločne a načas, kedže častokrát výsledok celého tímu je závislý od časovania jednotlivých jedincov, ktorých práca na seba nadväzuje.

8.3 Záver

Tímový projekt bol veľmi zaujímavý a aj keď sme mali spočiatku problém s pridelením témy, vďaka našej pedagogickej vedúcej sme sa v problematike rýchlo zorientovali a dovolíme si tvrdiť, že kvalita našej práce a výsledok vynaloženého úsilia rozhodne nezaostáva za konkurenčným tímom.

9 Zdroje

[1] CPN Tools: http://wiki.daimi.au.dk/cpntools/_home.wiki

[2] Petri .NET Simulator: http://www.petrinetsimulator.com/

[3] TimeNET: http://pdv.cs.tu-berlin.de/~timenet/

[4] Wikipedia.org – Petri Net: http://en.wikipedia.org/wiki/Petri_net

[5] Wikipedia – AGV: http://en.wikipedia.org/wiki/Automated_Guided_Vehicle

[6] Milan Češka, Vladimír Marek, Petr Novosad, Tomáš Vojnar: *Petriho Síte, PES, Studijný opora*. Ustav inteligentních systémů, Fakulta informačních technologiií, VUT v Brně.

[7] Tadao Murata: *Petri Nets: Properties, Analysis and Applications*

[8] UPPAAL: http://www.uppaal.com/

[9] Wikipedia.org – Dijkstrov algoritmus: http://sk.wikipedia.org/wiki/Dijkstrov_algoritmus

[10] A branch-and-bound algorithm for flow-path design of automated guided vehicle systems : http://www3.interscience.wiley.com/journal/113392086/abstract?CRETRY=1&SRETRY=0

[11] Bc. Dušan Ertl, Ing. Peter Drozd: Petriho siete a ich využitie v pri analýze správania sa modelu výrobnej linky - http://www.urpi.fei.stuba.sk/files/folders/docs/284_ERTEL.pdf

[12] Ing. Ján Liguš: Petriho siete - http://poprad.fei.tuke.sk/~ligus/irs/PNs.pdf