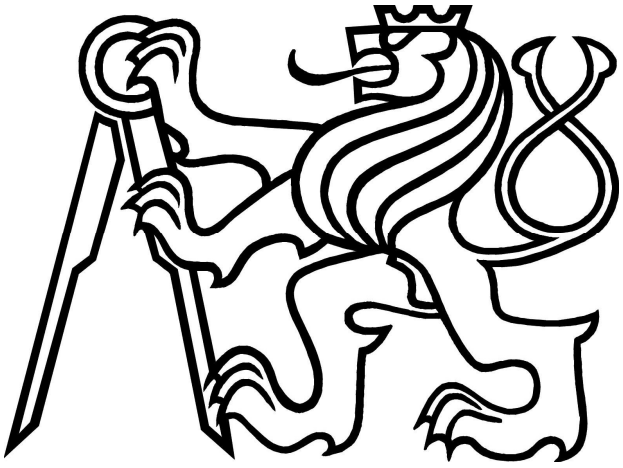
České vysoké učení technické
Fakulta elektrotechnická
Katedra řídicí techniky



Bakalářská práce

Simulátor výrobní linky

Jan Herzán

Vedoucí práce: Prof. Dr. Ing. Zdeněk Hanzálek

Studijní program: Softwarové technologie a management, Bakalářský

Obor: Inteligentní systémy

23. května 2013

Zadání

Poděkování

Rád bych zde poděkoval panu Prof. Dr. Ing Zdeňku Hanzálkovi a Ing. Romanu Čapkovi za jejich rady a čas, který mi věnovali při řešení dané problematiky.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne 23.5.2013 .....

Abstrakt

Tato práce je věnována problematice počítačové simulace výrobních linek. Práce se nejprve zabývá analýzou obecné výrobní linky a jejich parametrů. Na základě této analýzy je navržena metodika simulování obecné výrobní linky. Pro vytvoření simulátoru bylo použito matematického aparátu Petriho sítí. Za pomoci této sítě umí simulátor realisticky napodobovat vlastnosti linky. Navržený přístup je implementován v jazyce C#. Dále práce obsahuje výsledky testování pro jednu výrobní linku a vyhodnocení dosažených výsledků.

Abstract

This thesis is dedicated to the issue of a computer simulation of the production line. First, it analyzes the generic production line and its parameters. Based on this analysis, the methodology of simulating the generic production line is proposed. The Petri net mathematical apparatus was used for the simulator creation. Using this net, the simulator can realistically mimic all the characteristics of the production line. The implementation is done in C# language. This text also contains the results of testing on a production line and the discussion of achieved results.

Obsah

Seznam obrázků

Seznam tabulek

Kapitola 1

1. Úvod

Se zvyšující se rychlostí doby rostou také konzumní nároky společnosti. Každý producent libovolného produktu se snaží pokrýt co největší poptávku na trhu. Nejenom zvyšující se poptávka je trend 21. století, ale také snaha o snižování ceny.

V tomto prostředí sériové výroby a výrobních linek je každá minuta drahá. Proto je pro zvyšování výkonu a propustnosti a tím snižování ceny, kladen velký důraz na hledání metody, které umožňují růst těchto vlastností linky.

Určitě existují spousty způsobů, které mohou pomáhat zvyšovat výkon výrobní linky. Každý však se liší cenou, dostupností a hlavně kvalitou výstupu.

Při empirickém pozorování výrobní linky můžeme nalézt jisté zákonitosti, které mohou ovlivňovat správnost a rychlost jejího chodu. Ale toto pozorování má své hranice. V takovou chvíli je zapotřebí hledat řešení více komplexní a které bude zároveň hledět na celý proces detailněji. Jako první se nabízí vytváření modelů.

Vytváření matematického modelu je sice jedna z variant, ale jeví se jako velmi komplikovaná. Závislosti mezi stroji jsou na to příliš složité.

Další možností je simulace, která probíhá rychle, bez jakéhokoliv zásahu do reálné linky, a je možné velice podrobně celý proces zkoumat. Simulátor je pro výrobní linku robustnější nástroj, než je matematický model.

Další výhodou simulátoru je fakt, že můžeme simulovat zařízení, která aktuálně nemáme k dispozici. Tudíž můžeme zjišťovat, které stroje je dobré vylepšit nebo dokoupit, aby se celkový výkon linky se zvýšil. Určitě se zde nabízí varianta, že simulace bude provedena před vystavěním samotné linky a následně bude poptávka po již konkrétních zařízeních s konkrétními vlastnostmi.

1.1 Cíle práce

Cílem této práce je nastudovat problematiku výrobní linky, navrhnout systém jak ji simulovat, následně tento systém naimplementovat a otestovat jeho funkčnost. Na závěr je důležité zhodnotit výsledky, zdali odpovídají reálné lince.

Kapitola 2

2 Nástin řešení

Jak již bylo řečeno, na začátku stojí pouze spousty údajů o lince. Souhrnně těmto údajům říkáme parametrizace. Z této parametrizace vytváříme virtuální linku pro náš simulátor, který umí simulovat systémy diskrétních událostí.

Protože výrobní linka je systém diskrétních událostí, bude na její simulaci potřeba silného matematického aparátu, který nám může poskytnout účinnou reprezentaci pro tyto systémy.

Systémy diskrétních událostí jsou velmi často popisovány stavovými automaty, přechodovou funkcí a aktuálním stavem. Pokud bychom však chtěli vytvořit z celé linky jeden stavový automat, pak by tento stavový automat nabral značně na velikosti, jelikož na lince může být velké množství zařízení. Navíc je zapotřebí dělit linku na jednotlivé systémy a subsystémy, což by dodalo k velikosti ještě velkou míru nepřehlednosti.

S přihlédnutím k rozsáhlosti celé linky se nám nabízí matematický nástroj Petriho síť. Za pomoci Petriho sítě můžeme kvalitně simulovat stavové automaty jednotlivých zařízení na lince a pohyb produktů po dopravnících.

Důležité je také, že výstup, který nám Petriho síť nabízí, bude velice dobře analyzovatelný.

2.1 Cíle řešení

Výsledkem práce bude nástroj, který umožní získat obraz toho, jak linka reaguje na změny bez zásahu do samotné linky. Tím, že simulátor umožní testovat různá nastavení linky, aniž by bylo zapotřebí na ni sáhnout, ušetří čas a tím i značnou míru prostředků. Na začátku stojí samotné nastavení a parametrizace linky. Konečným produktem by měl být průběh výkonů jednotlivých zařízení a výpis poruch jednotlivých zařízení. Celý proces, od prvotního zadání nastavení linky, až po konečné vytvoření výsledků, by měl být co nejjednodušší pro uživatele, který bude simulátor obsluhovat.

Samotný simulátor by měl vytvářet náhodné události (například poruchy strojů), které by měly dodávat simulaci reálnost. Tyto události by měl mít možnost vytvářet i uživatel a tím simulovat situace z minulosti.

Kapitola 3

3 Výrobní linka

Před samotnou simulací je důležité se seznámit s výrobní linkou. Simulátor je vytvářen pro libovolnou výrobní linku, tudíž budeme o věcech, které se po lince pohybují hovořit jako o produktech.

Výrobní linka je soustava propojených zařízení, která z elementárních surovin vytváří konečný produkt. Linka je sestavena ze strojů, které provádějí operace se surovinami nebo s produkty, a dopravníků, které dopravují suroviny nebo produkty mezi stroji a zároveň slouží jako akumulační systémy pro vyrovnání neplánovaných stopů některých zařízení.

3.1 Použité termíny

Před seznamování s jednotlivými elementy linky je třeba ujasnit některé pojmy, které se v této práci vyskytují.

3.1.1 Porucha

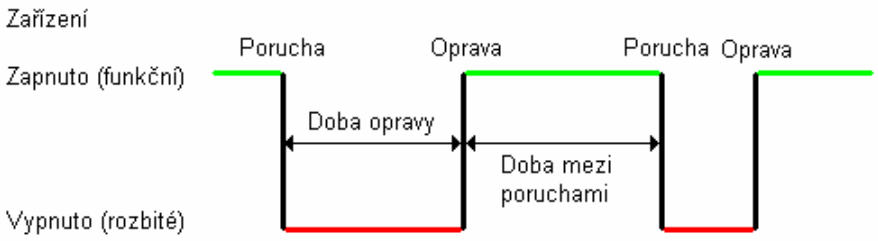
Poruchu můžeme definovat jako ukončení schopnosti zařízení jako celku vykonávat požadovanou funkci [1].

3.1.2 MTBF – Mean Time Between Failures (Střední doba mezi poruchami)

Střední dobou mezi poruchami je aritmetický průměr doby mezi poruchami. Dobou mezi poruchami označujeme čas, po který zařízení pracuje od zahájení jeho činnosti po poruchu (viz Obrázek 3.1) [2] .

3.1.3 MTTR – Mean Time To Repair (Střední doba opravy)

Střední doba opravy je aritmetický průměr doby opravy. Dobou opravy označujeme čas, který je potřeba k opravení zařízení (viz Obrázek 3.1).



Obrázek 3.1 - Doba mezi poruchami a doba opravy

3.2 Stroje

Celá linka se skládá z mnoha strojů, z nichž každý vykonává nějakou činnost spjatou s výrobou produktu. Stroje můžeme rozdělit do dvou základních kategorií.

První kategorií jsou stroje s deterministickou rychlostí. Deterministická rychlost znamená, že se po nastavení jejich kadence nemění.

Druhou skupinou jsou stroje, které deterministickou rychlost nemají. Jejich produkce osciluje v nějakém rozmezí. Pro simulaci těchto strojů je důležité vědět rozmezí, ve kterém se pohybují v jednotlivých stavech, ale i pravděpodobnostní rozložení rychlosti.

Všechny stroje mají několik společných parametrů. Těmito parametry jsou:

- MTBF - Střední doba mezi poruchami udávaná v sekundách [s]
- MTTR - Střední doba opravy udávaná v sekundách [s]
- Počet kusů produktů, na kterých aktuálně pracuje
- Parametry určující, zdali je zařízení generátor, případně konzument produktů. Pokud zařízení disponuje jednou nebo oběma z těchto vlastností, pak je nutné vědět i parametry pro tyto vlastnosti (frekvence generování nebo konzumace produktů).
- Parametry pro stavový automat (viz Stavový automat stroje)

3.2.1 Stavový automat stroje

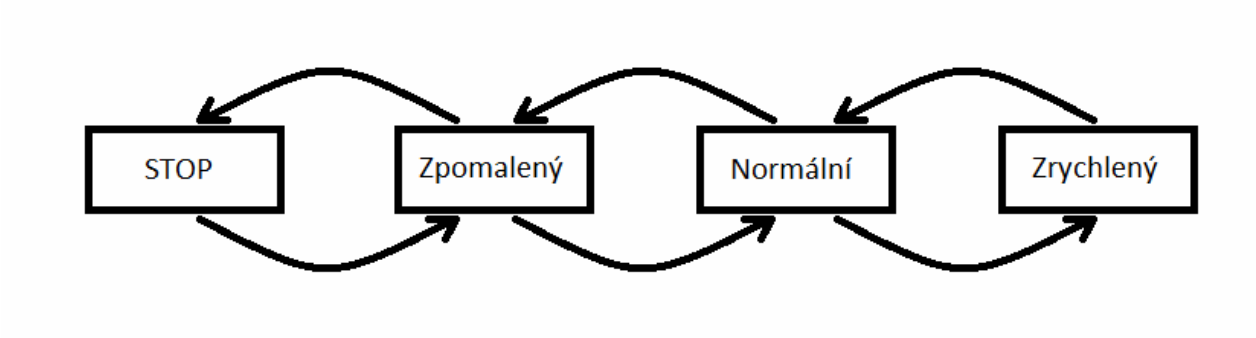
Stroj jako takový má dva stavové automaty, které se vzájemně ovlivňují.

Prvním je stavový automat, který reprezentuje vlastní poruchovost stroje, tedy zda je daný stroj aktuálně v poruše, nebo v provozním stavu. Tento stavový automat budeme nazývat stavový automat poruchy.

Druhým stavovým automatem je rychlostní stavový automat. Tento automat určuje, jakou rychlostí stroj aktuálně pracuje.

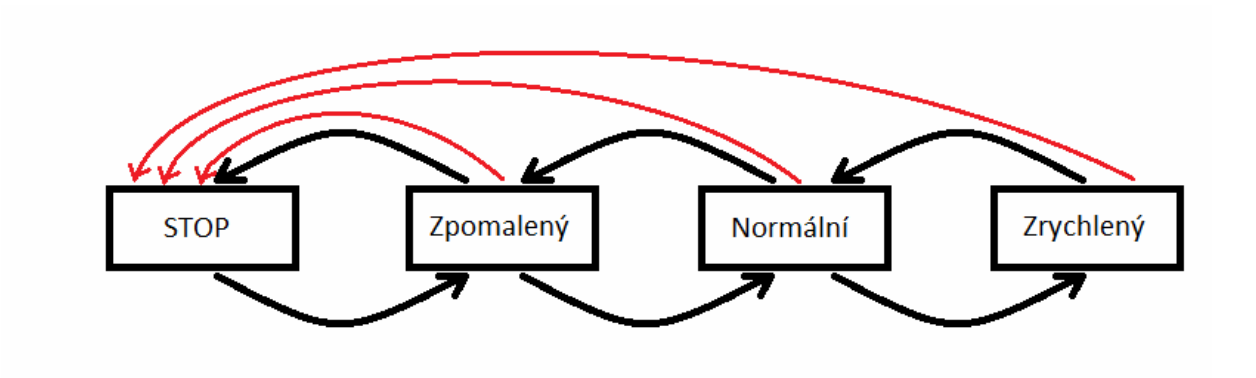
Rychlostní stavový automat stroje si lze představit jako řetěz stavů, které mají přechody pouze do sousedních stavů (viz Obrázek 3.2). Pokud se tedy chceme dostat ze stavu zpomalený do stavu zrychlený, je zapotřebí (protože spolu bezprostředně nesousedí) projít všemi stavy mezi nimi (v našem uvedeném případu stavem normální).

Každý ze stavů určuje, jakou rychlostí stroj běží [počet produktů za minutu].



Obrázek 3.2 - Rychlostní stavový automat stroje

Pouze do jediného stavu je možné se dostat okamžitě, a to je stav Stop, do kterého se můžeme dostat přechodem označovaným za Hard Stop. Tento přechod znázorňuje okamžité zastavení stroje z důvodu poruchy. Tento Hard Stop můžeme nalézt v reálné lince z bezpečnostních důvodů, aby se zamezilo dalšímu poškození následkem poruchy. Stavový automat s možností Hard Stop vidíme na Obrázku 3.3 vyznačené červenou barvou.



Obrázek 3.3 - Rychlostní stavový automat stroje s Hard Stopy

Stavový automat stroje je ovlivňován pouze stavovými automaty vedlejších zařízení, což jsou vždy dopravníky (dva stroje nesmějí spolu bezprostředně sousedit).

Rychlostní stavový automat stroje nemusí obsahovat všechny stavy, respektive nemusí obsahovat stav zpomalený nebo stav zrychlený. Stav stop a stav normální jsou definovány vždy.

3.2.2 Přechody mezi stavy stroje

Celá idea řízení je, že stroje se snaží stále držet ideální zaplnění dopravníků. Ve chvíli, kdy se dopravník před daným strojem začíná zaplňovat, musí stroj zvýšit výkon, aby nedošlo k jeho zaplnění a tím zastavení předcházejícího stroje. Pokud se však začne zaplňovat dopravník za daným strojem (například následkem toho, že následující stroj je porouchaný), je zapotřebí ulevit tomuto dopravníku a následujícímu stroji tím, že snížíme jeho rychlost.

Rychlostní stavový automat stroje reaguje na události, které zaznamenávají jeho vedlejší dopravníky (Předchozí a Následující). Oba tyto dopravníky poskytují stroji informace

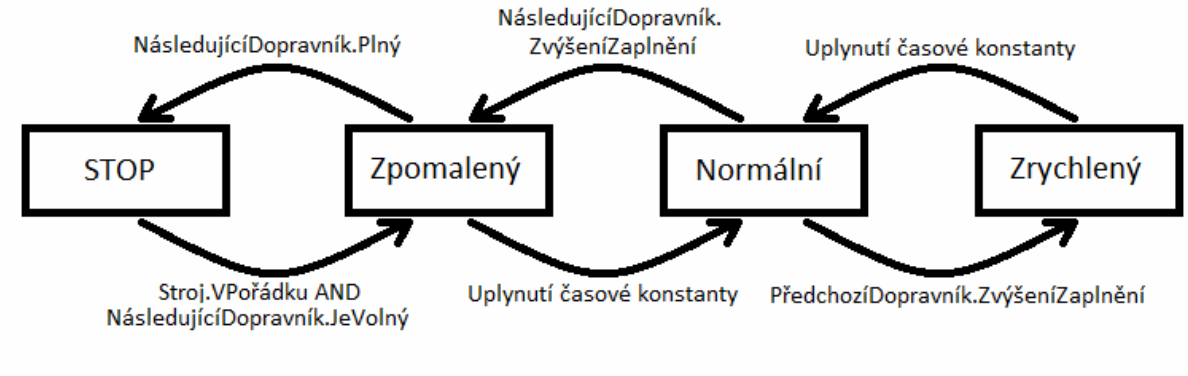
o jejich zaplněnosti. Tyto informace jsou předávány za pomocí proměnné, která může nabývat stavů:

- Plný - na dopravník se nevejdou žádné další produkty
- Zvýšení zaplnění - následující stroj neodebírá dostatečně produkty a ty se na dopravníku hromadí
- Volný - na dopravníku je dostatek místa pro další produkty

Změna následuje okamžitě ve chvíli, kdy přijde některý ze signálů z dopravníku. Celý stavový automat i s přechodovými podmínkami je vidět na Obrázku 3.4.

Změna stavu však nenastává, pokud stroj nemá na vstupu produkty, na kterých by mohl pracovat. Pak sice neprovádí žádnou činnost, ale svůj stav nemění. Ve chvíli, kdy se opět na předchozím dopravníku objeví produkty, zahajuje práci rychlostí, ve které byl.

Přechody mezi stavy nejsou skokové, ale pozvolné. Přestože to nejsou regulérní stavy, je zapotřebí s nimi počítat. Proto tyto přechodové fáze jsou vyobrazeny ve stavovém automatu jenom jako přechody a ne jako samostatné stavy. Tyto přechodové stavy mají dva parametry - rychlost stroje v daném přechodovém stavu a jak dlouho přechodový stav trvá.



Obrázek 3.4 - Rychlostní stavový automat stroje s přechodovými podmínkami

3.3 Dopravníky

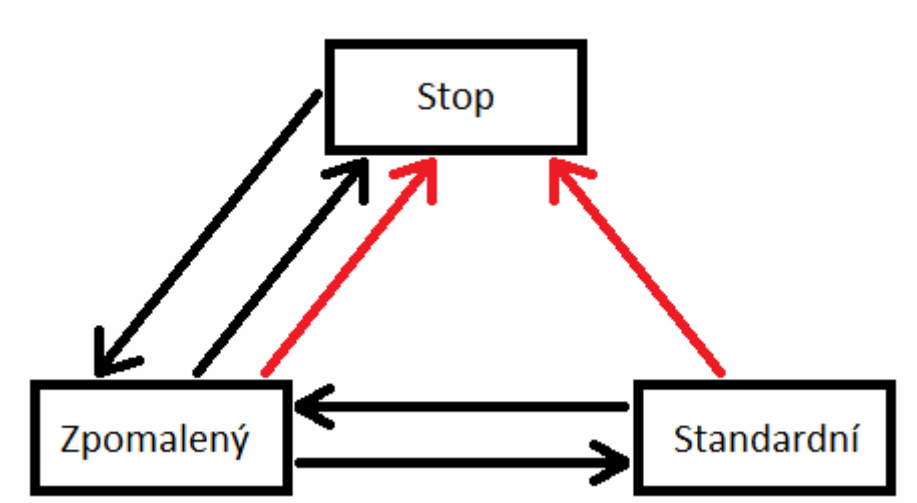
Dopravník je zařízení, které přepravuje suroviny nebo produkty mezi stroji. Pro jednoduchost si jej představme jako pohybující se pás, na kterém jsou vyskládány produkty v konstantním rozestupu od sebe. Pokud nastane porucha některého stroje, dopravník má za úkol zajistit kumulaci produktů, aby nedošlo ke zbytečnému zastavení ostatních strojů. Akumulace probíhá za pomoci zaplňování mezer mezi produkty, čímž se dočasně zvýší hustota produktů na dopravníku.

Stejně jako stroj i dopravník má několik základních parametrů:

- MTBF - Střední doba mezi poruchami udávána v sekundách [s]
- MTTR - Střední doba opravy udávána v sekundách [s]
- Délka dopravníku udávána v centimetrech [cm]
- Parametry pro stavový automat (rychlosti a doby přechodu mezi jednotlivými stavy)

3.3.1 Stavový automat dopravníku

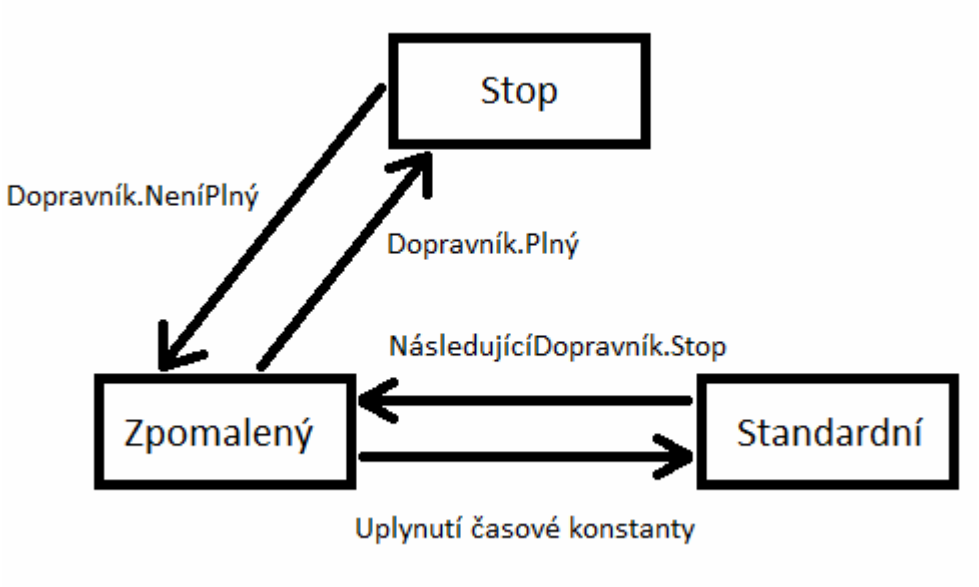
Stavový automat je pro všechny dopravníky stejný. Žádný ze stavů nemůže být opomenut, jako je tomu u stroje. Dopravník se může nacházet ve stavech: Stop, Standardní (Run), Zpomalený (Reduced). Stejně jako u stroje platí, že ve chvíli, kdy se má dopravník dostat do stavu Standardní ze stavu Stop, pak musí projít skrze stav Zpomalený (viz Obrázek 3.5). Dopravník má z bezpečnostních důvodů také Hard Stop, který je na Obrázku 3.5 vyznačen červenou šipkou.



Obrázek 3.5 - Stavový automat dopravníku

3.3.2 Přechody mezi stavy dopravníku

Stavový automat dopravníku se řídí svou vlastní zaplněností a dále stavovým automatem dopravníku, který ho následuje. Změna stavu nastává okamžitě, jakmile přijde patřičná informace z vedlejšího dopravníku.



Obrázek 3.6 - Stavový automat dopravníku s přechodovými podmínkami

Kapitola 4

4 Petriho síť

Základním stavebním kamenem simulátoru je Petriho síť. Do Petriho sítě bude přeložena celá výrobní linka a za pomoci vytvořené sítě může být simulována.

Základy Petriho sítě položil německý matematik Carl Adam Petri ve své disertační práci „Kommunikation mit automaten“ v roce 1962 [3]. Systém Petriho sítí byl mnohokrát rozšířen, aby dosáhl ještě lepších simulačních vlastností. Petriho sítě jsou využívány při modelování komunikačních protokolů, systémů diskrétních událostí, průmyslových řídicích systémů a mnoha dalších.

Petriho sít je orientovaný ohodnocený bipartitní graf, který má dvě základní skupiny uzlů, označovaných jako přechody a místa, spojených orientovanými ohodnocenými hranami.

Každé místo může obsahovat libovolný počet značek, které se nazývají tokeny.

Kdybychom hledali podobnost se stavovým automatem, pak si můžeme místa představit jako jednotlivé stavy a přechody jako události, které řídí změny stavu. Pozice tokenu může znázorňovat, ve kterém stavu se stavový automat aktuálně nachází.

Problematika Petriho sítí je probírána v [4,5], kde se autoři zaměřili na označené Petriho sítě.

4.1 Označená Petriho síť

Označená Petriho síť je uspořádaná pětice (P, T, Pre, $Post$, m_0), pro kterou platí:

- P je konečná množina míst (places), neboli $P = \{p_1, p_2, \ldots, p_K\}$, kdy K je celkový počet míst.
- T je konečná množina přechodů (transitions), neboli $T = \{t_1, t_2, \ldots, t_L\}$, kdy L je celkový počet přechodů.
- Pre je matice zobrazení $P \times T \rightarrow Z$. Každý prvek matice Pre_{ij} znázorňuje váhu hrany z místa i do přechodu j.
- $Post$ je matice zobrazení $P \times T \rightarrow Z$. Každý prvek matice $Post_{ij}$ znázorňuje váhu hrany z přechodu i do místa j.
- m_0 je vektor zobrazení $P \rightarrow Z$ reprezentující počáteční značení. Jinými slovy pro každé místo určuje, kolik tokenů obsahuje na počátku vývoje Petriho sítě.

4.2 Uvolnění a přeskok přechodu

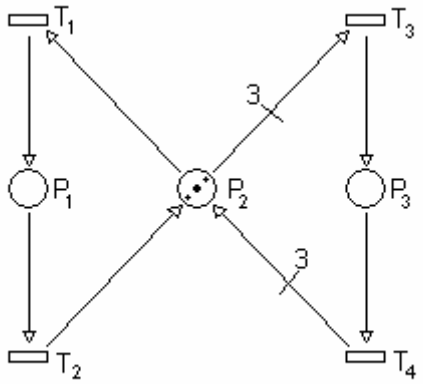
Libovolný přechod *T* je uvolněn právě tehdy, když všechna jeho vstupní místa (místa, ze kterých vede do *t* hrana) mají počet tokenů větší nebo roven váze hrany, která je spojuje.

Pokud je přechod uvolněn, může být přeskočen. Přeskočení přechodu odebere ze vstupních míst tokeny a přidá tokeny do míst výstupních. Počet tokenů odebraných, respektive přidaných, se řídí váhou hrany, která spojuje přechod s daným místem.

4.3 Příklad Petriho sítě

Pro lepší představu uvedeme příklad, který je velice příhodný pro svou jednoduchost.

Představme si nákladní nádraží, na kterém mají k dispozici 3 lokomotivy. Nabízejí služby odvozu malých zásilek, které uveze jedna lokomotiva, nebo velkých zásilek, na kterou jsou však zapotřebí všechny tři lokomotivy. Bystrý čtenář si rychle domyslí, že tedy můžeme najednou přepravovat maximálně tři malé zásilky, nebo jednu velkou.



Obrázek 4.1 - Ukázka Petriho sítě

Na Obrázku 4.1 je vidět model Petriho sítě, který znázorňuje výše popsaný problém. Máme zde tři místa, která představují, na jaké zakázce lokomotivy aktuálně pracují. Místo P_1 představuje práci mašiny na malé zásilce, místo P_2 představuje nepracující lokomotivy a místo P_3 představuje práci na velké zásilce.

Odjezd lokomotiv z depa, jinými slovy přiřazení lokomotiv k zásilkám, symbolizují přechody T_1 a T_3. Naopak příjezd zpět do depa je symbolizován přechody T_2 a T_4. Díky váze hran se při přeskoku přechodu T_1 spotřebuje pouze jeden token (odjede pouze jedna lokomotiva), čímž je zajištěno, že může tento přechod být přeskočen až třikrát po sobě. Na druhou stranu se zase při přeskoku přechodu T_3 odeberou tři tokeny, a tudíž může být přeskočen pouze jednou. To však může nastat pouze v případě, že v místu P_2 jsou k dispozici tři tokeny (všechny tři lokomotivy).

4.4 Konflikty

Síť má *Strukturální konflikt* právě ve chvíli, kdy máme dva výstupní přechody z jednoho místa. Takovýto konflikt může způsobit nedeterminismus v síti.

Síť má *Efektivní konflikt* právě ve chvíli, kdy strukturální konflikt nemá dostatek tokenů k přeskočení všech uvolněných přechodů. Příkladem efektivního konfliktu je výše uvedený případ rozložení lokomotiv.

4.5 Čas v Petriho sítích

Pokud chceme pracovat s časem uvnitř Petriho sítě, musíme uvést, že se vždy jedná o neautonomní Petriho síť. Neautonomní Petriho síť je podmíněna externími vstupy, což čas bezpochyby je.

Důležité je, že se rozšiřuje podmínka uvolnění přechodu o doběhnutí času daného přechodu.

Často se setkáváme se třemi různými modely.

Prvním modelem je *Časovaná Petriho síť*. V takovéto síti je každému přechodu přiřazen deterministický čas, za jaký může dojít k jeho přeskočení.

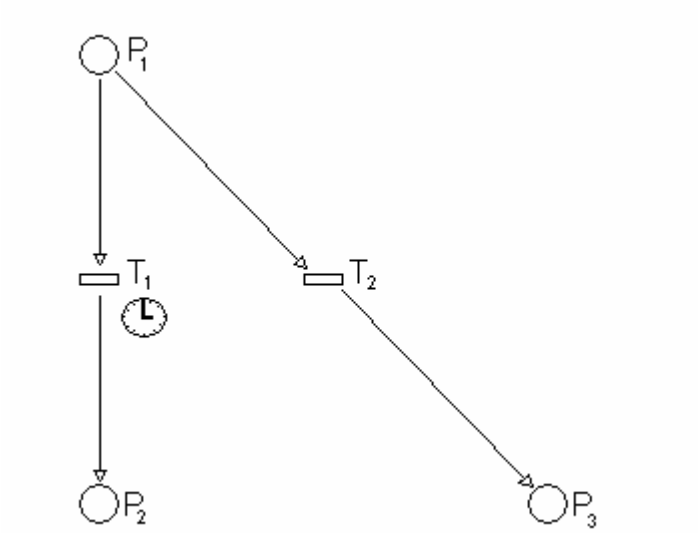
Druhým modelem je *Stochastická Petriho síť*. V takovéto síti má každý přechod svou funkci s pravděpodobnostním rozložením času.

Třetím modelem je *Časová Petriho síť*. V této síti má každý přechod dané rozmezí (horní a dolní mez) časového intervalu, za jaký může být přeskočen.

4.6 Časování přechodů

V předchozí kapitole bylo vysvětleno, jak se přiřazuje čas k přechodům. Důležité však je, kdy a jak se tento čas mění.

Představme si nejjednodušší situaci. Máme přechod T_1 mezi dvěma místy (viz Obrázek 4.2), který je spojen s časovačem. Ve chvíli, kdy se v místě P_1 objeví token, tento časovač postupně snižuje hodnotu času, která má v sobě uložený. Ve chvíli, kdy se časovač dostane na hodnotu nula, přechod se stává uvolněným a je možné ho přeskočit. Po přeskočení přechodu se hodnota časovače nastaví na počáteční hodnotu a celý proces může proběhnout znovu.



Obrázek 4.2 - Časování přechodů

Důležité však je, co se stane, když uprostřed snižování hodnoty v časovači, je token z P_1 odebrán důsledkem konfliktu - přeskoku jiného přechodu (označme jej T_2), který vychází též z místa P_1. Mohou nastat tři možné situace.

První situací je, že časovač v přechodu T_1 je resetován na původní hodnotu, stejně jako kdyby byl přeskočen. Takovýto přechod označujeme jako *resetující se přechod* nebo lépe *přechod s resetující se pamětí*.

Druhá možnost je, že se časovač v přechodu T_1 zastaví na hodnotě, na které aktuálně je, a odpočítávání bude pokračovat až ve chvíli, kdy přijde do místa P_1 nový token. Takovýto přechod označujeme jako *přechod se setrvávající pamětí*.

Poslední varianta je, že časovač odčítá čas nezávisle na přítomnosti tokenu v místě P_1.

4.7 Zdrojové místo

Zdrojové místo je speciální druh místa, do kterého nikdy nevede hrana. V tomto místě se nachází konstantní počet tokenů, který ani při odebírání neubývá. Toto místo se využívá pro simulaci zdroje.

4.8 Konzumní místo

Konzumní místo je speciální druh místa, ze kterého nikdy nevede hrana. Do tohoto místa mohou tokeny pouze vstupovat. Konzumní místo se využívá pro simulaci spotřebiče.

Kapitola 5

5 Token player

Token player je nástroj, pomocí kterého můžeme pohybovat tokeny uvnitř Petriho sítě. Úkolem Token playeru je upravovat čas v přechodech, kontrolovat jejich uvolněnost a následně je přeskakovat. Další věcí je, že musí zaznamenávat, který přechod byl v daný čas přeskočen.

Vstupem pro Token player je Petriho síť s počátečním rozložením tokenů. Toto může být definováno za pomoci uspořádané pětice (P, T, Pre, $Post$, m_0) (viz kapitola 4 - **Petriho sítě**).

Výstupem je uspořádaná sekvence dvojic (čas, přechod), která říká, kdy byl který přechod přeskočen. Dále je výstupem konečné rozložení tokenů v síti m_1.

Celý algoritmus probíhá ve dvou vnořených smyčkách. Vnitřní smyčka stále kontroluje, zdali je některý z přechodů uvolněný a pokud je, pak tento přechod přeskočí. Vnější smyčka inkrementuje čas ve chvíli, kdy už nejsou žádné přechody uvolněné. Algoritmus končí ve chvíli, kdy čas přesáhne předem určenou hranici.

```
totalTime;  // proměnná obsahující, jak dlouho má celý algoritmus běžet
            // tato proměnná je udána jako externí parametr algoritmu
actualTime = 0; // aktuální čas pro token player a pro všechny prvky
                //Petriho sítě
petriNet;       // Objekt obsahující všechny prvky Petriho sítě

while (actualTime <= totalTime) {
   while (petriNet.hasFireableTransition()) {
         // Dokud je některý přechod uvolněný
         petriNet.GetFireableTransition().Fire();
         // Přeskočí uvolněný přechod
   }
   actualTime.increaseTime();  // navýší čas
                               // (nemusí vždy být navýšen pouze o jedna)
}
```

Pseudokód Token playeru

Kapitola 6

6 Vytvoření modelu v Petriho síti

V této kapitole se budeme zabývat tím, jak vytvořit Petriho síť, za pomoci níž budeme simulovat výrobní linku.

Celé vytváření sítě probíhá v cyklu, při kterém se napojují nová zařízení k již vytvořenému modelu. Vzniká tím řetízek zařízení, které jsou vzájemně propojeny, aby mezi nimi mohly probíhat interakce.

Využití Petriho sítí v simulacích je popsáno v [6], kde se autoři zabývají výrobními systémy.

6.1 Množina sdílených míst α

Pro propojení jednotlivých zařízení je zapotřebí nějakého rozhraní, skrze které si budou tato zařízení předávat informace. Pro tyto účely zavádíme *množinu sdílených míst α*, která bude obsahovat místa, která budou sloužit k tomuto přenosu. Přenos informace bude probíhat za pomoci přítomnosti, nebo nepřítomnosti tokenu v místě.

Tato množina obsahuje místa:

- Signál, že na předchozím dopravníku se zvyšuje zaplnění
- Signál, že následující dopravník je plný
- Signál, že na následujícím dopravníku se zvyšuje zaplnění
- Signál, že následující dopravník je volný
- Signál, že následující dopravník je ve stavu Stop
- Místo předávající produkty
- Místo konzumující prázdné místo (více o tématice přesouvání produktů po lince v kapitole Překlad dopravníku)

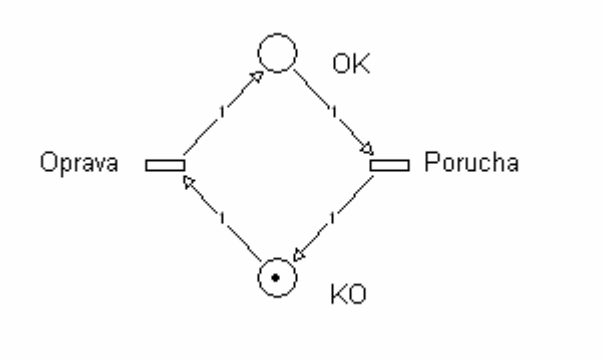
Protože linka nemůže začínat dopravníkem, musíme pro první stroj vytvořit umělou množinu míst α. Tato množina je vytvořena následovně:

- Místo indikující zvýšení zaplnění předchozího dopravníku je bez tokenu, protože předchozí dopravník se nikdy nezaplní, jelikož tam není. Toto místo je označeno jako zdrojové místo.
- Místa pro následující dopravník jsou vytvořena stejně, jako při připojování nového dopravníku (viz Překlad dopravníku).
- Místo předávající produkty je označenou jako Zdrojové místo – z tohoto místa budou proudit všechny produkty (v podobě tokenů) na lince.

- Místo konzumující prázdné místo je označeno jako Konzumní místo – toto místo musí být označeno jako konzumní, protože z něj nikdy nepovede hrana.

6.2 Napojení stroje

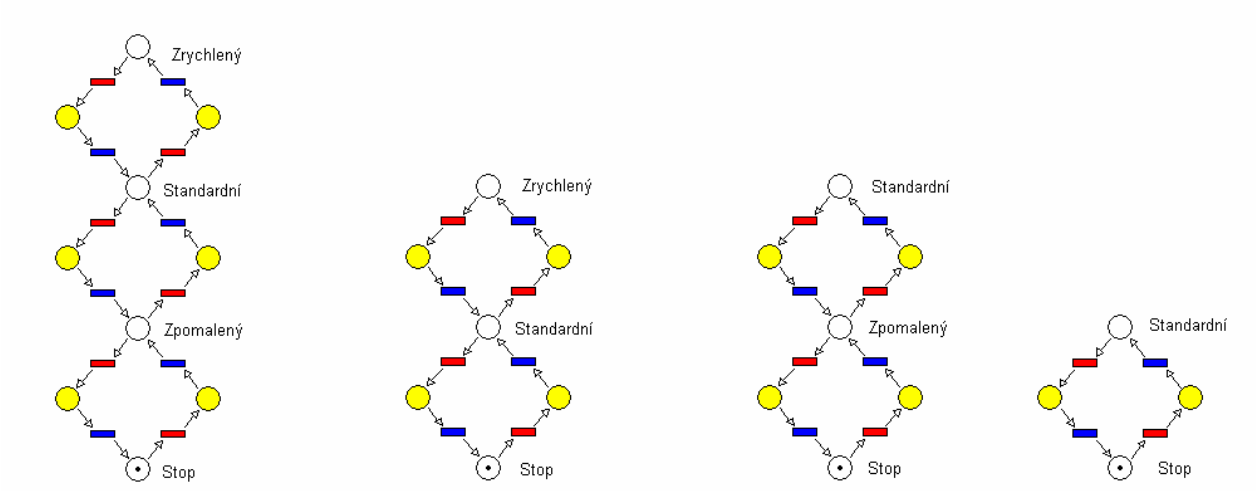
Jako první věc, která se vytváří při napojování nového stroje, je automat, který určuje, zda je daný stroj připravený, nebo v poruše. Dále ho budeme nazývat stavový automat poruchy. Petriho síť reprezentující tento stavový automat je na Obrázku 6.1.



Obrázek 6.1 - rozbitý/v pořádku stavový automat

Jako druhá věc se vytváří rychlostní stavový automat stroje, který určuje aktuální rychlost chodu stroje. Tento automat obsahuje 2-4 základní stavy - stop, zpomalený (nepovinný), standardní, zrychlený (nepovinný). Dále obsahuje přechodové děje reprezentující přechody mezi těmito stavy. Tyto mezi-stavy jsou zde z důvodu lepšího přiblížení se realitě, jelikož reálný stroj nezmění svoji rychlost skokově, ale je to (sice krátký) proces.

Všechny typy stavových automatů jsou vyobrazeny na Obrázku 6.2. Přechody, které odpovídají přechod do jiného stavu, jsou vyznačeny červenou barvou. Modrou barvou jsou označeny přechody, které určují dobu setrvání v mezi-stavu. Žlutě jsou označena místa, která znamenají přechodový děj.



Obrázek 6.2 - Stavové automaty stroje

Pro další vyobrazování, budu již používat pouze stavový automat, který obsahuje všechny stavy. Velmi důležité je místo reprezentující stav stop, protože je v něm jeden token. Tento token symbolizuje, v jakém stavu se aktuálně stavový automat nachází. Celý tento automat musí být pro účel simulace zabezpečen tak, aby se v něm nacházel právě jeden token. Tento token je během vytváření a na počátku simulace ve stavu Stop.

Při vytváření těchto stavových automatů je zapotřebí výše zmíněné množiny míst α. Z této množiny čerpáme místa, která jsou propojovací a ovlivňují řízení stroje. Dále rychlostní stavový automat ovlivňuje stavový automat poruchy. Jejich propojení je vidět na Obrázku 6.3.

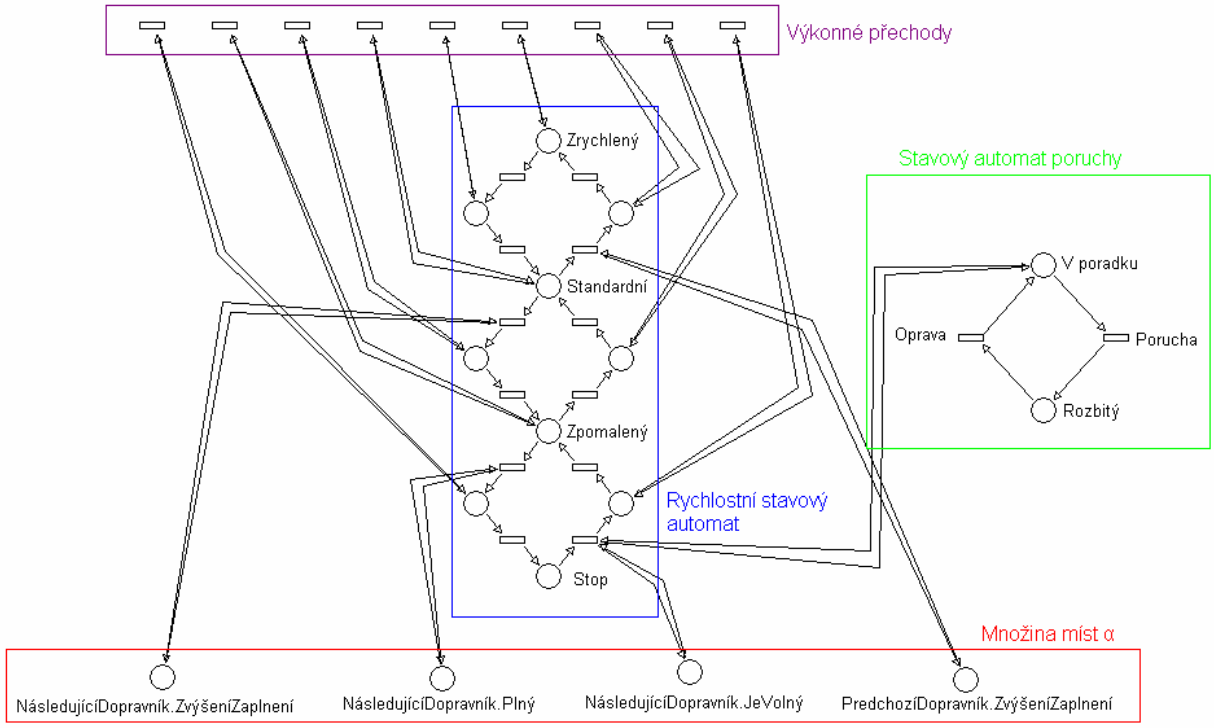
Další věcí je připojení Hard Stop přechodů, které ze všech stavů i mezi-stavů způsobí okamžité zastavení stroje. Tyto přechody na Obrázku 6.3 nejsou, protože by jej učinily nepřehledným.

Nyní je náš celý stavový automat stroje hotov. Teď je zapotřebí z každého stavu vyvést přechod, který symbolizuje průchod produktu strojem. Tyto přechody budeme označovat jako výkonné přechody. Tyto přechody jsou znázorněny na Obrázku 6.3.

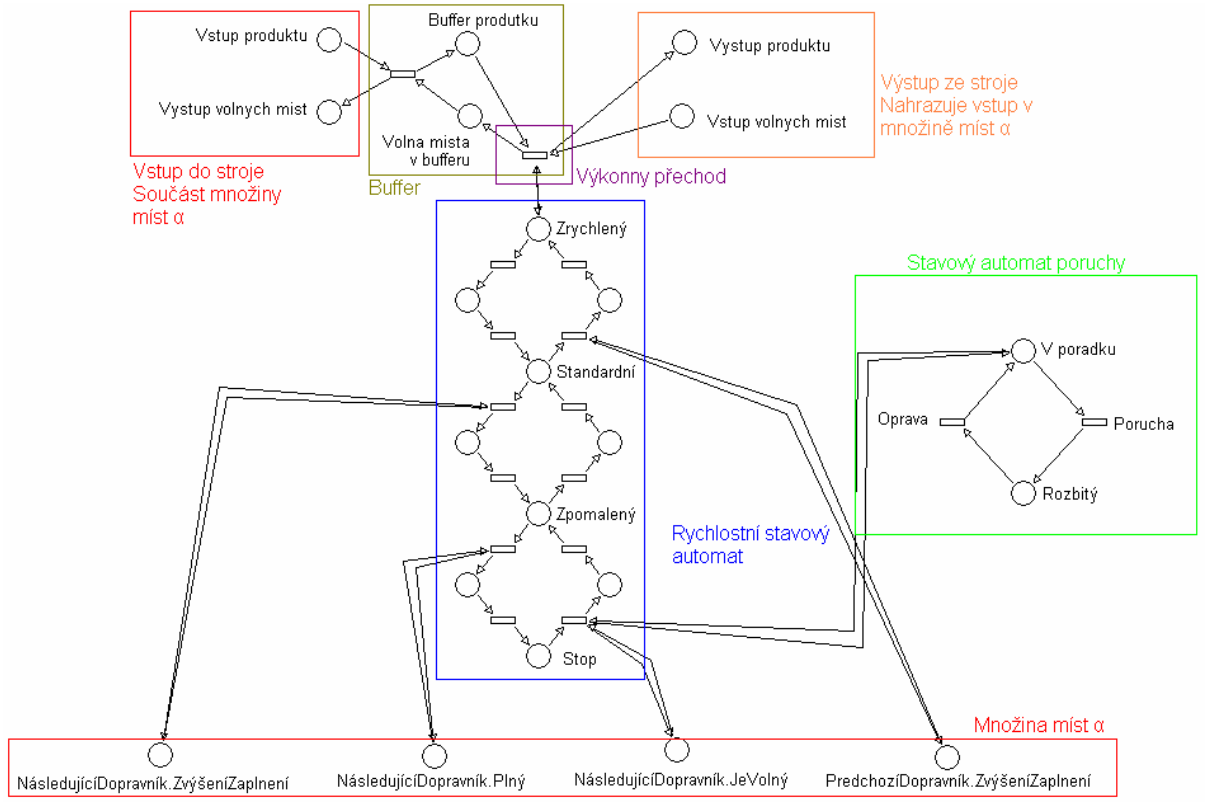
Další věcí, která je nutná pro vytvoření sítě, je poznatek, že stroj nepracuje pouze na jednom produktu. Stroj může pracovat paralelně na předem určeném počtu produktů a je nutné v síti znázornit, že tyto produkty se aktuálně nenacházejí na dopravnících, ale v samotném stroji. Tudíž než připojíme stroj k sousedním dopravníkům, je zapotřebí vytvořit zásobník (tzv. buffer), který bude hromadit tyto produkty nacházející se ve stroji.

Když se pozorně podíváme na Obrázek 6.4 níže, pak z pozice bufferu, který se nachází PŘED výkonnými přechody, pak zjistíme, že přeskok tohoto přechodu znázorňuje vystoupení produktu ze stroje. Dále na Obrázku 6.4 je již vyobrazené připojení k vedlejším dopravníkům.

Pro přehlednost byly všechny výkonné přechody nahrazeny jedním přechodem, který jde ze stavu zrychlený.



Obrázek 6.3 - Petriho síť stavového automatu stroje



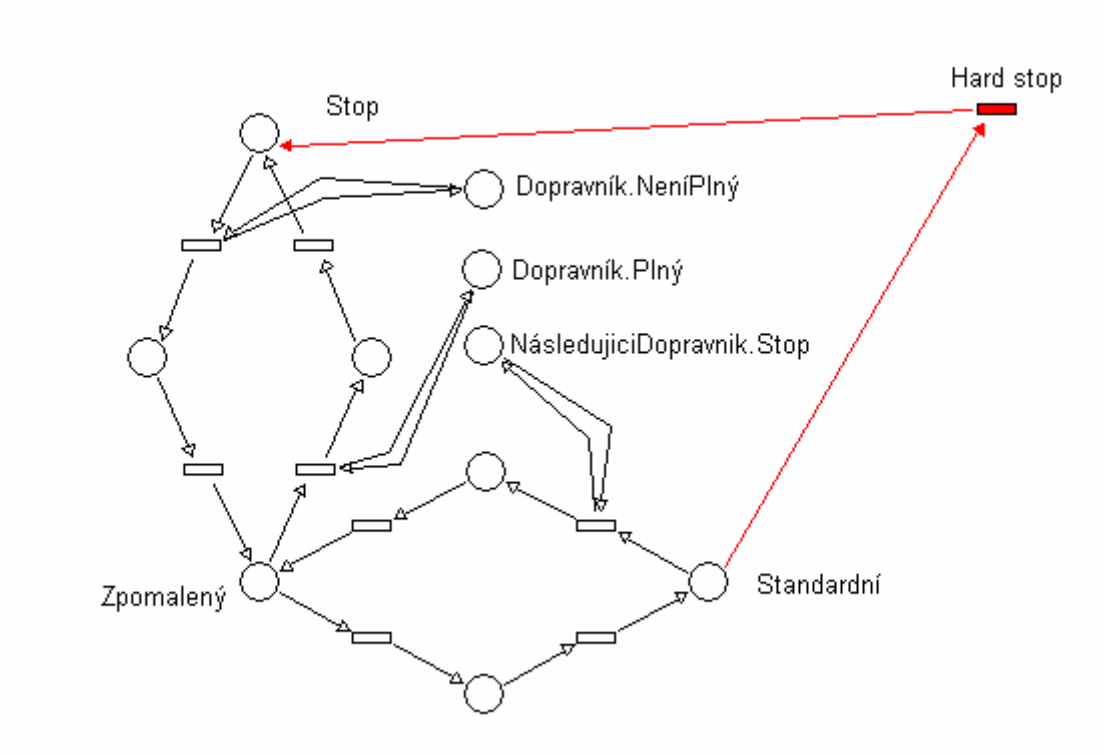
Obrázek 6.4 - Petriho síť stavového automatu s bufferem

Nyní je Petriho síť pro nový stroj hotová. Teď už stačí jenom změnit místa v množině míst α, aby bylo možné napojit další zařízení. Množina α se při připojení stroje změní pouze ve dvou věcech a to, že vstupní místa do stroje se nahradí výstupními místy ze stroje.
Nahrazení je vidět na předchozím Obrázku 6.4.

6.3 Připojení nového dopravníku

Jak již bylo výše zmíněno, stavový automat je pro všechny dopravníky stejný, žádný stav nesmí být opomenut.

Při připojování dopravníku se jako první vytváří síť symbolizující stavový automat, ale jsou k tomu využity místa, která jsou předávána za pomoci množiny míst α. Stavový automat dopravníku je vidět na Obrázku 6.5.

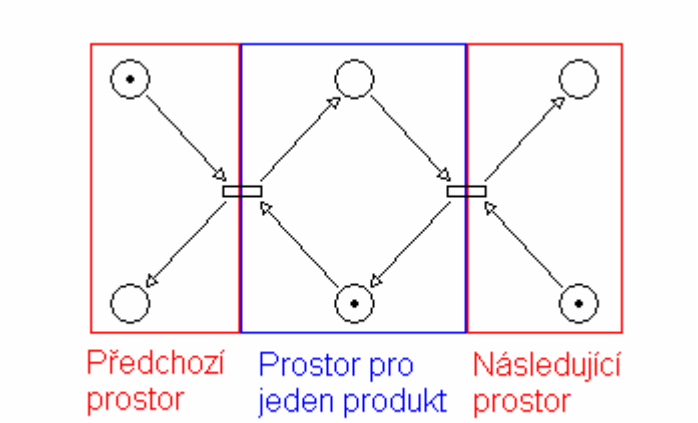


Obrázek 6.5 - Stavový automat dopravníku

Při vytváření "pásu", po kterém se přesouvají produkty, je důležitá idea reprezentace.

Protože Petriho síť nám nabízí pouze diskrétní události, musíme diskretizovat pohyb produktu po dopravníku.

Rozdělíme tedy dopravník na úseky, které budeme nazývat prostor pro jeden produkt. Tento prostor se skládá ze dvou míst a dvou přechodů. Na obrázku 6.6 je vidět jeden prostor pro jeden produkt se dvěma sousedícími prostory.



Obrázek 6.6 - Prostor pro jeden produkt

V této malinké síti se může pohybovat jenom jeden token. Pokud se nachází ve vrchním místě, pak se v daném prostoru nachází produkt (na Obrázku 6.6 je to vidět u levého prostoru). Pokud se však v daném prostoru produkt nenachází, token je ve spodním místě (tento případ je na Obrázku 6.6 vyznačen u prostředního a pravého prostoru). Tento způsob převodu má dvě základní výhody. První výhodou je, že v průběhu simulace je vždy vidět, jak moc a hlavně, v jakých částech je dopravník zaplněný. Druhou výhodou je, že pokud se začne dopravník zaplňovat z důvodu nedostatečného odběru následujícího zařízení, pak se produkty seskupují (pás pod nimi prokluzuje). Tato vlastnost je pro naši linku a tudíž i pro naši sít žádoucí.

Při vytvoření pásu se tedy vytvoří patřičný počet prostorů pro produkty a následně se spojí se stavovým automatem.

Závěrečnou fází připojování dopravníku je změna množiny míst α:

- Veškerá místa, která byla označena jako pro dopravník následující, jsou přeoznačena jako pro dopravník předchozí.
- Místa posledního prostoru pro produkty se stávají místy v rozhraní mezi zařízeními.
- Vytvářejí se nová místa pro následující dopravník (indikace jeho zaplněnosti).
- Vytvářejí se nová místa pro stavový automat následujícího dopravníku. Token se nachází v místě stavu Stop.

Při připojení všech zařízení je linka kompletní. Při vytváření sítě se již umisťují tokeny tak, aby nebyla potřeba další úprava před zahájením simulace. Linka je tedy hotová a připravená k simulaci.

Kapitola 7

7 Simulátor

Simulátorem můžeme označit nástroj, který na začátku dostane parametry linky a na konci vykreslí průběhy rychlostí jednotlivých strojů, zaplněnosti dopravníku apod.

V případu simulátoru, který je popisovaný v této práci, na začátku stojí konfigurační soubor. Tento konfigurační soubor obsahuje veškeré parametry linky, kterou budeme chtít simulovat (tyto parametry a jejich pořadí je možné nalézt v příloze 1). První úloha simulátoru je z tohoto konfiguračního souboru vytvořit objektovou reprezentaci výrobní linky a přiřadit jednotlivé parametry těmto objektům. Tuto objektovou podobu linky budeme označovat jako Device model.

Druhá věc, která se provádí ve chvíli, kdy je Device model hotový, je překlad do Petriho sítě. Celý proces probíhá ve smyčce, která bere jednotlivá zařízení a připojuje je k již vytvořené lince. Jediné spojovací rozhraní mezi zařízeními je objekt, který reprezentuje množinu míst α. To, jak simulátor překládá jednotlivá zařízení, je podrobně popsáno v kapitole 6 - **Vytvoření modelu Petriho sítě**.

Po vytvoření Petriho sítě, je celá tato síť simulována za pomocí Token playeru. Ten dostává jako parametr délku simulace a vzorkovací periodu. Popis Token playeru je v kapitole 5 - **Token player**.

Výstup z Token playeru je zapotřebí zpětně analyzovat, jelikož Token player, sám o sobě nerozumí tomu, co simuluje. Jinými slovy, Token player neví, co které místo znamená ani co reprezentují jednotlivé přechody, pouze simuluje podle pravidel Petriho sítí.

Analýza výstupu z Token playeru začíná tím, že se vyhledají určité přechody, které reprezentují změnu stavu stroje, změnu stavu dopravníku a pohyb produktů po lince. Tyto přechody jsou spjaté s Device modelem, ve kterém je možné nalézt jakou rychlost stroj má (v patřičných jednotkách). Průběh rychlosti je vykreslen do souborů, které jsou pojmenovány po zařízení, jehož graf byl vykreslen.

Dále je vytvořen soubor, ve kterém jsou vypsána celková trvání jednotlivých stavů pro každý stroj.

Kapitola 8

8 Implementace

Hlavním úkolem celého software je vytvoření výrobní linky, její simulace a následné vyhodnocení výsledků ze simulace. Celý program je rozdělen do tří základních knihoven a to do ***DeviceLib.dll***, ***Simulation.dll*** a ***TokenPlayerLib.dll***. Dělení knihoven je navrženo tak, aby každá spravovala jednu část simulace a byla nahraditelná jiným nástrojem.

DeviceLib.dll má za úkol spravovat informace o výrobní lince. ***Simulation.dll*** má za úkol samotnou simulaci, to znamená překlad modelu z ***DeviceLib.dll*** do modelu pro ***TokenPlayer.dll*** a následnou analýzu výstupu z ***TokenPlayer.dll***.

Implementace je v jazyce C#.

Poznámka autora: Pro lepší orientaci v dokumentu jsou barevně odlišeny ***Knihovny*** *(*.dll),* ***Rozhraní*** *(interface),* ***Třídy*** *(class) a* ***Výčtové typy*** *(Enum).*

8.1 DeviceLib.dll

Tato knihovna slouží k vytvoření samotného modelu linky. Knihovna obsahuje objektovou reprezentaci jednotlivých zařízení. Dále též se stará o načítání z externího konfiguračního souboru a o kontrolu její konzistence.

Knihovna je rozdělena do tří základních částí. První částí je složka **Models**, která obsahuje stavební kameny linky. Druhou částí je složka **Providers**, která obsahuje nástroje pro práci s linkou. A třetí částí je složka **Exceptions**, která obsahuje výjimky, které můžou být vyhozeny v průběhu práce s linkou.

8.1.1 Models

Složka obsahující veškeré objekty, ze kterých se může linka sestavovat. Tato část knihovny obsahuje vše potřebné pro vytvoření/editaci/práci s výrobní linkou (resp. její programovou reprezentací).

8.1.1.1 IDevice.cs

Základní interface pro veškerá zařízení tvořící linku. Je nutné, aby každé zařízení bylo jednoznačně rozpoznatelné a tudíž je mu přiřazeno *ID* a *name* (jméno). ID si musí hlídat uživatel knihovny sám, jediné, co knihovna hlídá sama, je duplicita ID.

Pro účely spojování a tvoření posloupnosti obsahuje ukazatel (objekt ***IDevice.cs***) na zařízení před sebou a na zařízení za sebou (upstream a downstream).

Posledními parametry jsou střední doba mezi poruchami a střední doba opravy zařízení. Obě dvě veličiny jsou zadávány v sekundách.

8.1.1.2 IMachine.cs

Tento interface slouží k práci s veškerými stroji na lince tzn. každý musí implementovat toto rozhraní.

Interface definuje vlastnosti, zdali je stroj generátor (*IsProducer*) nebo konzument (*IsConsument*) produktů. Generátorem je myšleno, že v něm produkty „vznikají“. Konzument je analogicky, že v něm produkty „zanikají“. A dále obsahuje celočíselné vlastnosti, které udávají, jaká je frekvence produkování/konzumování produtků - *RatioOfConsumedProducts* [ppm – počet produktů za minutu] a *FrequencyOfProducedProducts* [ppm – počet produktů za minutu].

Dále stroj obsahuje parametr, který udává, kolik se do něj vejde produktů.

Důležitý prvek, který musí každý stroj obsahovat, je stavový automat, který je též dostupný přes toto rozhraní. Prvky tohoto automatu jsou popsány níže *(viz **StateModel.cs**)*.

8.1.1.3 IConveyor.cs

Rozhraní, které musí být implementováno každým objektem symbolizujícím dopravník.

Každý dopravník obsahuje svou délku, která je udávána v centimetrech.

Stejně jako stroj, dopravník má také stavový automat, který je z tohoto rozhraní dostupný. Prvky tohoto automatu jsou popsány níže (*viz **StateModel.cs***).

8.1.1.4 StateModel.cs

Tato třída obsahuje jednotlivé stavy stavového automatu a přechody mezi nimi. Tato třída je generická, což znamená, že je použitelná jak pro stavový automat dopravníku, tak pro stavový automat stroje. Jako generický typ může být pouze ***MachineStateType*** nebo ***ConveyorStateType***.

Samotné stavy jsou reprezentovány pomocí objektů třídy ***SpeedDefinition.cs***, která je popsána níže.

Každý přechod ze stavu do stavu obsahuje parametry:

- dobu přechodu z jednoho stavu do druhého [sekundy]
- časovou konstantu, která určuje, jak dlouho má stavový automat setrvat v následujícím stavu (tím myšleno ve stavu, do kterého přecházím tímto přechodem) [sekundy]

- koeficient rychlosti, kterým se násobí rychlost vyššího stavu, aby byla dosažena rychlost přechodu z jednoho stavu do druhého. Vyšším stavem je myšleno stav, který má vyšší rychlost. Důvodem užívání vyššího stavu je, aby dané číslo bylo vždy menší než jedna.

8.1.1.5 MachineStateType.cs

Tento výčtový typ obsahuje všechny stavy stavového automatu, do kterých se může stroj dostat.

- *breakDown*
  - Stroj stojí
- *reducedRun*
  - Stroj běží ve zpomaleném módu (v módu se sníženou rychlostí)
- *standardRun*
  - Stroj běží v módu s prioritní rychlostí
- *speedupRun*
  - Stroj běží ve zrychleném módu (v módu se zvýšenou rychlostí)

8.1.1.6 ConveyorStateType.cs

Tento výčtový typ obsahuje všechny stavy stavového automatu, do kterých se může dopravník dostat.

- *stop*
  - Dopravník se nepohybuje
- *run*
  - Dopravník se pohybuje plnou rychlostí
- *reduced*
  - Dopravník se pohybuje redukovanou rychlostí

8.1.1.7 SpeedDefinition.cs

Třída obsahuje veškeré informace o rychlosti stroje. Objekt této třídy obsahuje hodnoty:

- *Nominal speed* (Nominální rychlost) [ppm – počet produktů za minutu]
  - rychlost pro daný stav
- *MinimalSpeedCoefficient* (Koeficient pro maximální rychlost)
  - když tento koeficient vynásobíme nominální rychlostí, dostaneme maximální rychlost, kterou v daném stavu může stroj dosáhnout

- *MaximalSpeedCoefficient* (Koeficient pro minimální rychlost)
    - když tento koeficient vynásobíme nominální rychlostí, dostaneme minimální rychlost, kterou v daném stavu může stroj dosáhnout

Koeficient pro minimální a maximální rychlost se určuje pro stroje, respektive jejich stavy, které nemají konstantní hodnotu (nejsou deterministické). U stroje s konstantní kadencí jsou implicitně nastaveny na 1.

8.1.1.8 Conveyor.cs

Konkrétní třída dopravníku implementující rozhraní ***IConveyor.cs***. Vytvořením instance této třídy získáváme objekt, do kterého ukládáme všechny o dopravníku známé údaje. Třída samotná obsahuje, kromě vlastností a metod definovaných v rozhraní ***IConveyor.cs***, kontrolní mechanizmy pro:

- délka dopravníku nesmí být záporná
- hodnoty MTBF a MTTR nesmí být záporné

Všechny tyto kontrolní mechanizmy při nesplnění podmínky vyhazují ***InvalidDeviceDefinitionException.cs*** (*viz* Exceptions) s patřičným popiskem chyby.

8.1.1.9 Machine.cs

Konkrétní třída stroje implementující rozhraní ***IMachine.cs***. Vytvořením instance této třídy získáváme objekt, do kterého ukládáme všechny o stroj známe údaje. Třída samotná obsahuje kontrolní mechanizmy pro:

- hodnoty MTBF a MTTR nesmí být záporné
- Nemohou být za sebe napojené dva stroje, stroj může být napojený pouze na dopravník a to jak zepředu, tak zezadu

Všechny tyto kontrolní mechanizmy při nesplnění podmínky vyhazují ***InvalidDeviceDefinitionException.cs*** (*viz* Exceptions) s patřičným popiskem chyby.

8.1.1.10 ProducitonLine.cs

Třída zapouzdřující celou plnící linku. Obsahuje parametr *productDiameter* (rozměr produktu), který nesmí být záporný, jinak je vyhozena ***InvalidProductionLineDefinitionException.cs*** (*viz* Exceptions). Dále obsahuje veškerá zařízení, která se v lince nachází. Jednotlivé stroje a dopravníky jsou do linky přidávány pomocí metod *AddMachine* a *AddConveyor*.

Před jakýmkoliv pracováním s objektem této třídy od někudy přijatým je zapotřebí, aby byla ověřena jeho konzistence za pomoci metody *CheckConsistency*.

8.1.2 Providers

Zde jsou veškeré třídy, které jsou schopné zpracovávat objekt ***ProductionLine.cs***.

8.1.2.1 IProductionLineProvider.cs

Toto rozhraní musí implementovat každá třída, která slouží k načítání výrobní linky ze vstupního souboru ***ProducitionLine.cs***.

Pro vytvoření instance nové linky je potřeba zavolat metodu *LoadProductionLine* s parametrem umístění zdrojového souboru.

Pro získání instance vytvořené linky je potřeba zavolat metodu *GetProductionLine*, která vrátí instanci vytvořené linky.

8.1.2.2 ProductionLineProvider.cs

Konkrétní implementace třídy, která načítá novou linku ze souboru. Při načítání souboru se zpracovávají jednotlivá zařízení tak, jak jsou psána v konfiguračním souboru, vytváří se jejich parametrizovaný objekt a řadí se za sebe do výrobní linky.

Konfigurační soubor pro tento provider je popsán v příloze číslo 1.

8.1.3 Exceptions

Tato složka obsahuje veškeré výjimky, které mohou nastat v průběhu práce s knihovnou ***DeviceLib.dll***.

8.1.3.1 InconsistentDeviceFileContentException.cs

Výjimka vyhazovaná při nekonzistentním obsahu řádku konfiguračního souboru. Obsahuje vždy pozici špatně načítaného řádku.

8.1.3.2 InvalidDeviceDefinitionException.cs

Výjimka vyhazovaná při přidání nekonzistentního parametru do objektů reprezentujících zařízení. Obsahuje vždy konkrétní popis, proč parametr nesplňuje podmínky.

8.1.3.3 InvalidProducitionLineDefinitionException.cs

Výjimka vyhazovaná, pokud není linka konzistentní z jakéhokoliv důvodu. Obsahuje konkrétní popis, proč linka není konzistentní.

8.2 TokenPlayerLib.dll

Tato knihovna obsahuje kompletní funkčnost Petriho sítí. Složka **Models** obsahuje prvky, za pomoci nichž lze vytvořit novou instanci samotné sítě, tuto instanci naplnit přechody, místy a hranami. Nástroje pro samotnou simulaci nalezneme ve složce **Player**.

Ve složce **Providers** můžeme najít potřebné nástroje pro formátování a ukládání výstupu ze samotného ***TokenPlayer.cs***.

8.2.1 Models

V této složce nalezneme veškeré objekty, které jsou zapotřebí k sestavení sítě jako takové.

8.2.1.1 Place.cs

Tato třída reprezentuje místo v Petriho síti.

Základními údaji, které jsou potřeba pro vytvoření místa, jsou:

- *Id* – ID
    - Třída sama o sobě žádným způsobem nekontroluje konzistenci ID
- *Name* – Jméno
    - Jméno jako takové je dobré k orientaci v chybových hláškách, kde je uváděno v případě chyby. Pokud není udáno, je vytvořené z ID
- *NumberOfTokens* – počet tokenů
    - Proměnná obsahující aktuální počet tokenů v místě
    - Při prvotní inicializace se nastavuje proměnou v konstruktoru *initialNumberOfTokens*
- *MaximalNumberOfTokens* – maximální počet tokenů v místě
    - Samotná proměnná je pouze pro kontrolu
    - Třída sama kontroluje, jestli nebyl překročen maximální počet tokenů v místě

Třída jako taková obsahuje metody pro kontrolu tokenů *addTokens* a *removeTokens* a pro manipulaci s hranami (***Arc.cs***) *addInputArc* a *addOutputArc*, které jsou do místa napojené. Pro správnou manipulaci s tokeny v místě je lepší užívat fasádu nabízenou třídou ***Arc.cs***.

8.2.1.2 SourcePlace.cs

Potomek třídy ***Place.cs***. Toto místo nemůže mít vstupní hrany, a tudíž do něj není možné vkládat tokeny, jelikož je zdrojové. V tomto místě tokeny neubývají, hodnota stavu tokenů zůstává stále na počáteční hodnotě.

8.2.1.3 SinkPlace.cs

Potomek třídy ***Place.cs***. Toto místo nemůže mít výstupní hrany a tudíž z něj nelze odebírat tokeny, jelikož je koncové. V tomto místě se tokeny „požírají“. Místo si pamatuje, kolik „požralo“ tokenů.

8.2.1.4 Transition.cs

Tato třída reprezentuje přechod v Petriho síti.

Základními údaji, které jsou potřeba pro vytvoření přechodu, jsou:

- *Id* – ID
    - Třída sama o sobě žádným způsobem nekontroluje konzistenci ID
- *Name* – Jméno
    - Jméno jako takové je dobré k orientaci v chybových hláškách, kde je uváděno v případě chyby. Pokud není udáno, je vytvořené z ID
- *Priority* – Priorita
    - Priorita určuje, který z přechodů má být přeskočen, pokud nastává kolize. Pokud by nastala kolize dvou přechodů se stejnou prioritou, přeskočí se přechod, který byl řadícím algoritmem dán na vyšší místo v pořadníku
- *Type* – Typ přechodu
    - Vysvětlení jednotlivých přechodů *viz* ***TranstionType.cs***
- *Distribution* – Pravděpodobnostní rozdělení časů pro daný přechod
    - Toto rozdělení určuje časy do dalšího přeskoku. Více *viz* **Distributions**.***IDistribution.cs*** a jednotlivé rozložení v podložce **Distributions**.

Přechod jako takový může vracet informace o své přeskočitelnosti metodami *HasTokensForFire*, *IsFireable* a *GetTimeToNextFire*.

Při samotném přeskočení přechodu se automaticky dá pokyn všem hranám (Arc.cs), aby přesunuly tokeny mezi místy, nastaví se nový čas do dalšího přeskoku a vyvolá se událost *Fired*, která je zachytávána samotným token playerem (*viz* **Player**.***TokenPlayer.cs***). Tato metoda kompletně spravuje přeskok v celé sítí, jinými slovy není potřeba nijak jinak dávat pokyny hranám, nebo místům, aby pohybovaly tokeny.

Metodou *Tick* dáváme přechodu pokyn o inkrementaci interního času. Tento interní čas musí být ve všech instancích jedné sítě stejný.

8.2.1.5 TransitionType.cs

Tento výčtový typ určuje, jakého typu bude daný přechod. Může nabývat hodnot:

- *restartTransition*
  - Tento přechod si restartuje svůj vnitřní čas při nedostatku tokenů na vstupu
- *continueTransition*
  - Tento přechod dekrementuje čas zbývající do přeskoku pouze tehdy, pokud má na všech vstupech dostatek tokenů.
- *alwaysTickingTransition*
  - Tento přechod žádným způsobem nekontroluje počet vstupních tokenů. Prostě pokaždé dekrementuje čas zbývající do přeskoku, pokud je tento čas vyšší než nula.

Jedna síť může obsahovat všechny typy přechodů.

8.2.1.6 Arc.cs

Tato třída reprezentuje hranu v Petriho síti. Hrany z místa do přechodu i z přechodu do místa jsou reprezentovány touto třídou. Změna je pouze v parametrizaci.

Základními údaji, které jsou potřeba pro vytvoření hrany, jsou:

- *Place* – Místo
  - Místo, se kterým je hrana spojená
- *Transition* – Přechod
  - Přechod, se kterým je hrana spojená
- *PlaceToTransition*
  - Logická hodnota, zdali hrana vede z místa do přechodu, nebo z přechodu do místa
- *weight* – násobnost hrany
  - Tento koeficient určuje násobnost hrany, jinak řečeno pokud hrana vede z místa do přechodu, pak udává, kolik tokenů se naráz zkonzumuje. V opačném případě kolik se jich vytvoří.

Tato třída slouží jak fasáda pro místo (Place.cs). Třída poskytuje metodu *HasTokens*, kterou se můžeme dotázat, zdali místo spojené s touto hranou obsahuje dostatek tokenů. Pokud tato hrana vede z přechodu do místa, pak nemá význam tento dotaz provádět.

8.2.1.7 Network.cs

Tato třída reprezentuje celou Petriho síť. Zapouzdřuje prvky popsané výše, které se přidávají do sítě metodami *AddPlace*, *AddTransition*, *AddArcTransitionToPlace* a *AddeArcPlaceToTransition*. Je možné se zpětně dotazovat na jednotlivé objekty v síti za pomoci jejich *ID* a to metodami *GetPlace* a *GetTransition*.

Obsahuje metody pro přidávání prvků do sítě, dotazy na jejich počty a kontrolu konzistence sítě. Kontrola sítě je složena z jednotlivých dílčích kontrol:

- zdali všechny místa a přechody navzájem propojené jsou v síti
- kontrola násobnosti hran, zdali síť neobsahuje koncové přechody-
- jestli jsou všechna místa, která nemají vstupní hranu označena jako ***SourcePlace.cs***
- jestli jsou všechna místa, která nemají výstupní hranu označena jako ***SinkPlace.cs***.

Posledním nástrojem, kterým tato třída disponuje, je nalezení všech následujících přechodů pro libovolný přechod za pomoci metody *GetSuccessiveTransitions*. Jinými slovy nalezení všech přechodů, které by mohly být přeskočitelné na základě přeskoku námi vybraného přechodu.

8.2.1.8 SimulationRecord.cs

Třída reprezentující objekt ukládající průběh běhu sítě. Ukládají se sekvence dvojic [čas, přechod] za pomoci metody *AddRecord*. Nad touto sadou záznamů se lze dotazovat, kdy byl daný přechod přeskočen, nebo které přechody byly přeskočeny v daný čas – metody *GetTimesForTransition* a *GetFiredTransitionsInTime*. Časy všech záznamů si můžeme vyžádat za pomoci *GetTimesOfEvents*.

8.2.2 Distributions

V této složce jsou umístěny třídy, které reprezentují pravděpodobnostní rozložení pro generování čísel.

8.2.2.1 IDistribution.cs

Rozhraní, které musejí všechna rozložení implementovat. Povinná je pouze metoda *GetNextTime*, která vrací nově vygenerovaný čas do dalšího přeskoku přechodu na základě daného rozložení.

8.2.2.2 ConstantDistribution.cs

Konstantní rozložení

8.2.2.3 UniformDistribution.cs

Rovnoměrné rozložení.

8.2.3 Player

Tato složka obsahuje samotný token player, který zprostředkovává veškerou potřebnou funkcionalitu pro simulaci Petriho sítě.

8.2.3.1 TokenPlayer.cs

Třída provádějící simulaci Petriho sítě.

Pro spuštění simulace je zapotřebí:

- *Network* – Sít
    - Síť, která bude simulována
- *runType* – mód, ve kterém simulace poběží
    - Módy jsou více popsány v ***TokenPlayerRunType.cs***.

Nad token playerem můžeme volat dvě funkce.

Pomocí *RegistrerSimulationRecordWriter* přiřadíme ***ISimulationRecordWriter.cs***, kterým chceme zaznamenávat průběh simulace.

Druhá funkce je *Run*, která spustí simulaci. Jejím parametrem je délka simulace [počet ticků = čas/délka ticku].

Celý proces simulace probíhá v několika stále se opakujících cyklech.

Metoda *Run*, dokud aktuální čas je menší než celkový čas simulace, volá provedení jednoho kroku cyklu – metodu *ExecuteNextStep*.

Metoda *ExexcuteNextStep* si udržuje seznam přechodů, které jsou uvolněné. Z něj vybere ten s nejnižší prioritou a ten přeskočí. Následně obnoví seznam – přidá nově uvolněné přechody a odebere přechody, které naopak přestaly být uzavřené přeskokem posledního přechodu. Tento postup opakuje, dokud má nějaký uvolněný přechod. Ve chvíli, kdy již žádný přechod není uvolněný, nastane inkrementace času metodou *IncreaseTime*.

Metoda *IncreaseTime* zvyšuje aktuální čas na základě proměnné *runType*.

8.2.3.2 TokenPlayerRunType.cs

Token player může běžet ve dvou módech:

- *stepByStep* – Čas simulace se bude vždy inkrementovat o jedna
- *eventDriven* – Čas simulace se bude vždy inkrementovat o čas zbývající do nejbližšího přeskoku přechodu

8.2.4 Providers

8.2.4.1 ISimulationRecordWriter.cs

Tento interface musí implementovat třídy, které mohou zaznamenávat přeskoky přechodů a následně jej ukládat. Interface obsahuje metody *OnTransitionFire* a *SaveRecord.*

OnTransitionFire je metoda volaná handlery, která dostává jako parametr přeskočený přechod a čas přeskoku.

SaveRecord je metoda, která vytvoří soubor se záznamy, ze kterého se později dá načíst kompletní výstup z token playeru.

8.2.4.2 ISimulationRecordReader.cs

Tento interface musí implementovat každá třída, která patří do páru k třídám impementujícím ***ISimulationRecordWriter.cs*** a jsou schopné načítat svým druhem vytvořený ***SimulationRecord.cs***. Interface obsahuje metody *LoadRecord* a *GetSimulationRecord*.

8.2.4.3 SimulationRecordWriter.cs

Základní implementace třídy zajišťující ukládání záznamů o průběhu simulace sítě, k čemuž slouží metoda *OnTransitionFire*. Dále obsahuje metodu pro uložení veškerých záznamů do souboru *SaveRecord*.

8.2.4.4 SimulationRecordReader.cs

Třída do páru s ***SimulationRecordWriter.cs***. Je schopná načíst záznam o průběhu simulace sítě a spárovat se sítí.

8.2.4.5 NetworkProvider.cs

Třída schopná načítat Petriho síť z konfiguračního souboru. Třída načítá konfigurační soubory vytvořené programem PM edit. Soubory mají příponu *.RDP. Výstupem je objekt typu ***Network.cs***, který je možné vyžádat metodou *GetNetwork*. Tento nástroj je určený pouze pro testování token playeru. Pro vytváření sítí bude sloužit transformace z ***DeviceLib.dll*** objektů.

8.3 Simulation.dll

Tato knihovna je spojovacím článkem mezi ***DeviceLib.dll*** a ***TokenPlayerLib.dll***. Stará se o celkový běh programu. Obsahuje tři základní části. První je **Conversion**, která obsahuje nástroje pro konverze. Dále jsou zde nástroje pro analýzu výstupu z ***TokenPlayer.cs*** ve složce **Analysis**. Poslední složkou jsou **Exceptions**.

8.3.1 Conversion

Tato složka obsahuje nástroje pro konverzi z Device modelu do Petriho sítě.

8.3.1.1 IDeviceNetworkConverter.cs

Tento interface musí být implementován každou třídou, která má za úkol provádět konverzi z Device modelu do Petriho sítě. Ke spojení s modelem slouží metoda *AssignProductionLine* a následné vrácení sítě probíhá za pomoci metody *GetNetwork*.

Pro aktivaci samotné konverze je zapotřebí spustit metodu *ConvertToNetwork*, která jako parametr dostává vzorkovací periodu – jak dlouhý je jeden tick [s].

Pro třídy, které budou provádět analýzu výsledků, jsou zde metody:

- *GetInputTransitionsForMachineState* – Vrátí přechody, které provádějí přechod do zadaného stavu stavového automatu stroje
- *GetInputTransitionsForConveyorState* – Vrátí přechody, které provádějí přechod do zadaného stavu stavového automatu dopravníku
- *GetOuptuTransitionForMachineState* a *GetOuptuTransitionForConveyorState* – analogicky, akorát pro výstup z daného stavu
- *GetInputTransitions* – vrátí vstupní přechody do daného zařízení
- *GetOuptuTransitions* – vrátí výstupní přechody z daného zařízení

8.3.1.2 BrokeFineStateMachine.cs

Tato třída slouží pro předávání přechodů, míst a hran stavového automatu, který značí, zda-li je zařízení rozbité, nebo v pořádku. Celkem obsahuje 2 přechody a dvě místa (místa stavu „V pořádku“ a „Rozbitý“, přechody „Porucha“ a „Oprava“).

8.3.1.3 DeviceCreatingInterface.cs

Pro spojování zařízení je potřeba určitých mezních míst, na které je napojeno další zařízení. Pro předávání těchto míst na rozhraní slouží tato třída.

Obsahuje místa:

  - Indikátory zaplněnosti předchozího dopravníku
  - Indikátory zaplněnosti následujícího dopravníku
- Stavový automat následujícího dopravníku a zněj stavy
  - Stop
  - Not stop
  - Run
  - Reduced

- Místa pro předávání lahví
    - Místo pro lahve
    - Místo pro „prázdná místa na lahve“

8.3.1.4 DeviceNetworkConverter.cs

Tato třída fyzicky překládá Device model do Petriho sítě. Celý proces probíhá, že se načítá zařízení za zařízením a připojují se za sebe. Vzájemná propojení zařízení probíhá za pomoci ***DeviceCreatingInterface.cs***.

Při připojování stroje je první vytvořen stavový automat ***BrokeFineStateMachine.cs***, dále stavový automat zakládající se na jednotlivých stavech stroje z ***MachineStateType.cs***. Následně probíhá propojení těchto dvou stavových automatů a vytvoření všech „produkty-zpracujících“ přechodů. Na závěr probíhá propojení s předchozím zařízením a generování nového rozhraní pro následující zařízení.

Při připojování dopravníku se vytváří nejdříve stavový automat dopravníku zakládající se na ***ConveyorStateType.cs***. Následně se vytvoří všechna místa (jednotlivé segmenty dopravníku) kudy mohou procházet produkty. Poté se propojí se stavovým automatem. Posledním krokem je připojení k vedlejšímu zařízení a vytvoření rozhraní pro následující zařízení.

Metoda *ConvertNetwork* může vyhazovat výjimky ***DeviceNetworkConversionException.cs*** s patřičnou poznámkou.

8.3.1.5 Analysis

Tato složka obsahuje nástroje pro analýzu simulace. Dále jsou v této složce nástroje pro vizualizaci výsledků (kreslení grafů).

Základní údaje se uchovávají v třídách ***MachineAnalysis.cs***, ***ConveyorAnalysis.cs*** a ***AnalysisRecord.cs***. Třídy pro zpracování údajů jsou ***SimulationRecordAnalyzer.cs*** a ***SimulationRecordVizulaizer.cs***.

8.3.1.6 MachineAnalysis.cs

Tato třída slouží k zaznamenávání a seřazení změn stavů stroje. Změny stavů se přidávají za pomoci metody *AddStateChange*. Metdodou *GetTimeSpeedProfile* je poté možné získat celý vývoj stavu stroje (a jeho rychlostí) v čase.

Tato třída též může vypsat průměrnou rychlost stroje (*GetAverageSpeed*) a celkový čas strávený v každém stavu (*GetTotalStateTime*) .

8.3.1.7 ConveyorAnalysis.cs

Tato třída je ekvivalentní ***MachineAnalysis.cs*** s tím rozdílem, že je určená pro uchovávání dat o dopravnících.

8.3.1.8 AnalysisRecord.cs

MachineAnalysis.cs obsahuje informace o stavech pro jeden stroj. Abychom mohli zpracovávat všechny zařízení, je zapotřebí všechny ***MachineAnalysis.cs*** a ***ConveyorAnalysis.cs*** zapouzdřit do jediného objektu, kterým je je ***AnalysisRecord.cs***. Přidávání analýzy do objektu je možné pomocí metody *AddMachineAnalysi*/*AddConveyorAnalysiss* a zpětné vrácení je možné přes metodu *GetMachineAnalysis*/*GetConveyorAnalysis*.

8.3.1.9 SimulationRecordAnalyzer.cs

Tato třída analyzuje vývoj Petriho sítě, získany pomocí ***TokenPlayer.cs*** a následné výsledky uloží do ***AnalysisRecord.cs***. Celý proces analýzy výsledků je zahájen metodou *Analyze* a celá analýza je exportována do souboru pomocí metody *ExportAnalysis*. Pro každý stroj se v cílovém souboru objeví průměrná rychlost a časy setrvání v jednotlivých stavech.

8.3.1.10 SimulationRecordVizualizer.cs

Tato třída vykresluje grafy průběhů rychlostí v čase. Pro každý stroj je vytvořen separátní graf, který je uložen do obrázkového souboru. Tento obrázkový soubor je pojmenován na základě jména stroje.

Celá vizualizace je spouštěná metodou *Vizualize*, která dostává jako parametry hotovou analýzu.

8.3.2 Exceptions

Tato složka obsahuje výjimky, které mohou být vyhozeny při běhu konverze, nebo analýzy.

8.3.2.1 DeviceNetworkConversionException.cs

Tato výjimka je vyhozena v průběhu konverze z Device modelu do Petriho sítě. Vyhození proběhne, pokud v průběhu konverze je nalezeno neznámé zařízení, nebo poslední zařízení je dopravník.

8.3.2.2 SimulationException.cs

Obecná výjimka, která může být vyhozena v průběhu simulace.

Kapitola 9

9 Testování a použití

V této kapitole se budeme zabývat testováním a využitím simulačního nástroje, který je popsán v kapitole 7 - **Simulátor**.

9.1 Testování jednotlivých knihoven

9.1.1 Test Token Playeru

Pro otestování bylo zapotřebí najít nějaký nástroj, který by byl schopen dávat stejný výstup, jako Token player popisovaný v této práci. K tomuto testování byl použit Matlab Toolbox for Petri nets [7]. Vstupem obou nástrojů byl textový soubor *.RDP, který obsahoval kompletní definici Petriho sítě.

Vývoj třídy TokenPlayer.cs (viz kapitola 7 - **Implementace**) potom probíhal s ohledem na to, aby výstup simulace pro 10 testovacích sítí odpovídal výstupu simulace získaného pomocí Matlab Toolboxu. Po jakékoliv změně ve třídě TokenPlayer.cs vždy došlo k opětovnému srovnání výsledků. Tímto byla zaručena konzistence funkčnosti s již dostupnými nástroji.

9.2 Použití simulačního nástroje

Pro ukázku funkčnosti simulačního nástroje byla vytvořena linka, na které je možné demonstrovat, jak simulátor pracuje.

Linka slouží k plnění limonády do prázdných lahviček. Máme zde šest zařízení, která postupně berou jednu lahvičku po druhé a pracují na ní. Těmito stroji na lince jsou vyndavač lahví z krabic, vymývač lahví, plnič lahví, zátkovačka, etiketovačka a krabicovač.

Provedením několika simulací můžeme demonstrovat rozdíl ve výstupu na základě rozdílných parametrizací linky.

9.2.1 Simulace 1-3

První simulace bude probíhat na lince, která má tyto parametry:

Pro všechny stroje platí:

- Jejich buffer (maximální počet lahví uvnitř stroje) je 16 lahví.
- Mean Time To Repair (MTTR) mají 500 sekund.
- Rychlost stroje ve standardním stavu je 100 lahví za minutu.

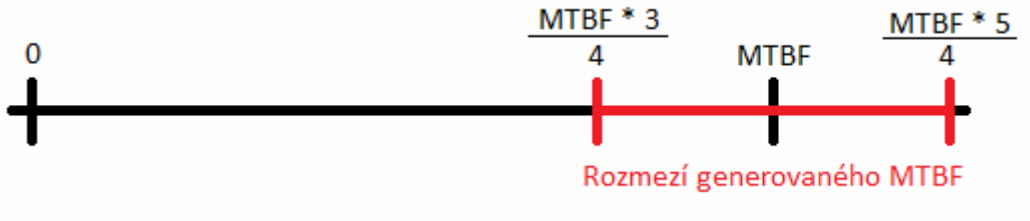
- Změna rychlosti při zrychleném a zpomaleném chodu stroje je 6%. Jinými slovy pro zrychlený stav je rychlost 106 lahví za minutu a pro zpomalený je 94 lahví za minutu.

Pro všechny dopravníky platí, že ve standardním stavu běží rychlostí 21 metrů za minutu a ve zpomaleném běží rychlostí 12 metrů za minutu

Hodnoty MTBF pro jednotlivé stroje:

- Vyndavač lahví – 2760
- Vymývač lahví – 4497
- Plnič lahví – 3693
- Zátkovačka – 2894
- Etiketovačka – 4220
- Krabicovač – 3907

Pro generování poruch byl využit generátor náhodných čísel s rovnoměrným rozložením a to vždy v rozsahu od tří čtvrtin do pěti čtvrtin MTBF každého stroje (viz Obrázek 9.1).



Obrázek 9.1 - Generování MTBF

Pro simulace 1-3 bude jediný rozdíl a to v délce dopravníku. Jak již bylo uvedeno v kapitole 3.3 - **Dopravníky**, dopravník nemá funkci pouze přepravovat produkty od jednoho stroje k druhému, ale i částečně vyrovnávat výkyvy produkce strojů.

V Tabulce 9.1 níže jsou uvedené některé parametry, které vychází z analýzy. Konkrétně je to délka dopravníků, čas simulace, průměrná rychlost jednotlivých strojů a časy, kdy byl stroj bez vstupních surovin. Celá simulace byla nastavena tak, aby odpovídala třem hodinám reálného běhu linky.

Výpočet průměrné rychlosti linky vychází ze vzorce:

$$\bar{v} = \frac{A}{t_s - t_{pl}}$$

$\bar{v}$ je průměrná rychlost stroje,

A je počet zpracovaných lahví za dobu simulace,

t_s je celkový čas simulace,

t_{pl} je doba do příjezdu první lahve (čas zahájení první práce).

Název proměnné	Simulace 1	Simulace 2	Simulace 3
Délka dopravníků	5 m	8 m	11 m
Čas simulace	488 s	669 s	1074 s
Průměrná rychlost linky	44,35 bpm	47,59 bpm	48,62 bpm
Čas bez surovin vyndavače lahví	0 s	0 s	0 s
Čas bez surovin vymývače lahví	00:20:45	00:19:20	00:17:26
Čas bez surovin plniče lahví	00:19:21	00:21:41	00:18:05
Čas bez surovin zátkovačky	00:38:06	00:25:09	00:29:34
Čas bez surovin etiketovačky	00:59:29	00:49:28	00:44:43
Čas bez surovin krabicovače	01:01:08	00:53:45	00:51:16

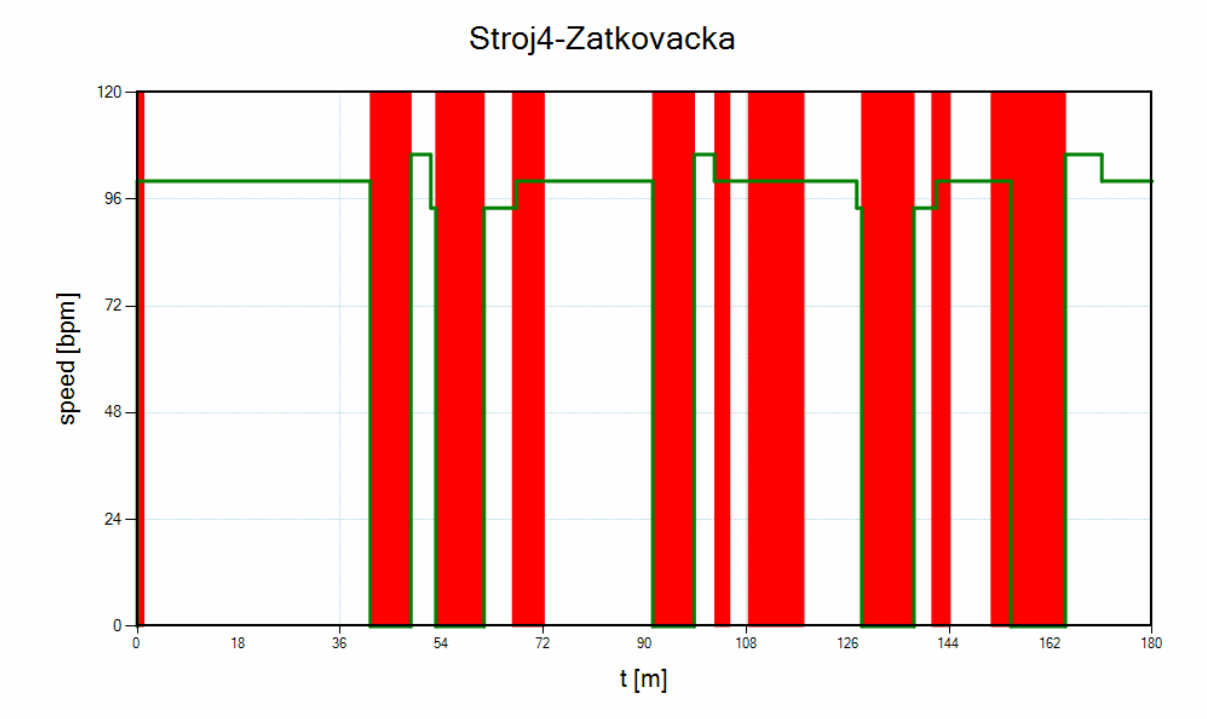
Tabulka 9.1 - Výsledky simulací 1-3

Je několik věcí, které jsou zřejmé z tabulky výsledků. První věc se týká samotného simulátoru. Prodloužení všech dopravníků o 60% mělo za následek zvýšení doby simulace o 37%. Při prodloužení dopravníků o 120% došlo k prodloužení simulace o 122%. Z toho je možné usoudit, že délka dopravníků hraje velkou roli v délce simulace.

Druhá věc, která je velmi dobře znatelná z výsledků, je, že akumulace produktů na dopravníku může udržovat linku déle v provozu při poruše některého ze stroje. Při porovnání první a druhé simulace je tento rozdíl velice znatelný. Třetí simulace je na druhou stranu důkazem toho, že nekonečné prodlužování dopravníků nemusí vždy vést k tak markantnímu zlepšení.

Následující Obrázek 9.2 je ukázka výstupu vizualizace ze simulace. Tento obrázek je ze simulace číslo 2. Zelená křivka znázorňuje rychlost stroje v čase. Červené prostory značí, že stroj nezpracovával lahve a to z jednoho ze dvou důvodů. Prvním důvodem může být, že se

nacházel v poruše a tudíž je červené místo tam, kde je rychlost stroje nulová. Druhým důvodem může být nedostatek lahví na vstupu.



Obrázek 9.2 - Ukázka výstupu ze simulátoru ze simulace 2

Z vizualizace průběhu Zátkovačky je vidět, že se stroj poté, co se opraví, snaží opět dostat dopravník před sebou do rovnovážného stavu. Na vizualizaci se to projeví tím, že se po opravě okamžitě přepne do stavu Zrychlený, a až ve chvíli, kdy se sníží zaplněnost předchozího dopravníku na normální hodnotu, pokračuje ve standardním chodu.

9.2.2 Simulace 4-6

V první sérii simulací bylo předvedeno, jakým způsobem bude ovlivněna propustnost linky, když se změní velikost dopravníků a tím i jejich akumulační schopnost. V druhé sérii si ukážeme, že zvýšení rychlosti všech strojů nemusí vést vždy ke zvýšení propustností linky.

Důležitý je fakt, že se zvyšující se rychlostí stroje roste jeho chybovost, tudíž Mean Time Between Failures (MTBF) se snižuje. Tato skutečnost vychází z empirického pozorování linky.

Pro druhou sérii simulací využijeme stejnou linku jako v prvním případě, až na několik malinkých změn. První změnou je, že délku dopravníku ponecháme osm metrů pro všechny simulace. Druhou věcí je, že budeme zvyšovat rychlosti stroje. Rozdíl rychlostí bude 10% a 20%. MTBF se však bude zvyšovat o 30% s každým zvýšením rychlosti. Všechny změny popisuje Tabulka 9.2 níže.

	Simulace 4	Simulace 5	Simulace 6
Délka dopravníků	8 m	8 m	8 m
Rychlost strojů ve zpomaleném stavu	94 bpm	103 bpm	112 bpm
Rychlost strojů ve Standardním stavu	100 bpm	110 bpm	120 bpm
Rychlost strojů ve zrychleném stavu	106 bmp	117 bpm	128 bpm
MTBF oproti Simulaci 1	100%	70%	40%

Tabulka 9.2 - Parametry linky pro simulace 4-6

Generování MTBF i výpočet průměrné rychlosti stroje je stejný jako v předchozí simulaci. Celá simulace byla nastavena tak, aby odpovídala třem hodinám reálného běhu linky. V Tabulce 9.3 níže jsou uvedené některé parametry, které vycházejí z analýzy. Konkrétně je to průměrná rychlost linky, délka simulace a Stop time jednotlivých strojů. Stop time je čas, jak dlouho byl stroj ve stavu Stop.

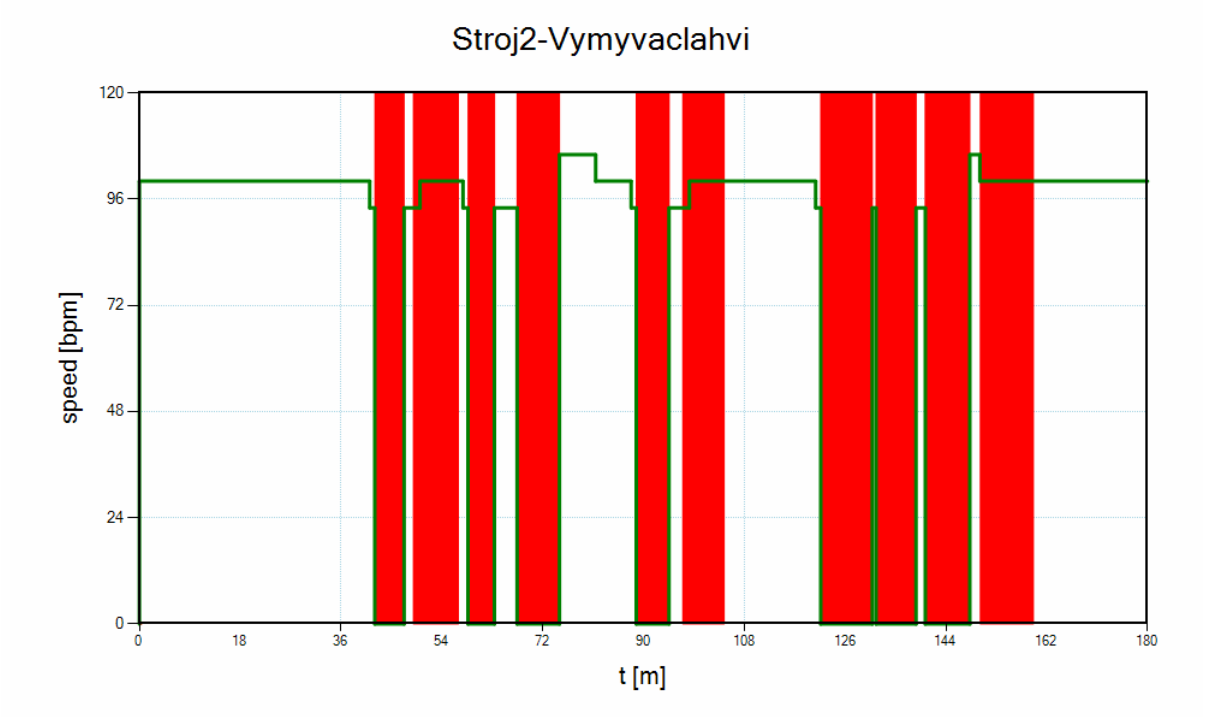
Název proměnné	Simulace 4	Simulace 5	Simulace 6
Čas simulace	705 s	710 s	709 s
Průměrná rychlost linky	44,69 bpm	45,32 bpm	30,82 bpm
Stop time vyndavače lahví	01:11:36 s	01:16:02 s	01:51:29 s
Stop time vymývače lahví	00:47:42	01:04:01	01:51:16
Stop time plniče lahví	00:37:11	01:00:39	01:36:41
Stop time zátkovačky	00:34:45	00:41:38	01:14:34
Stop time etiketovačky	00:18:31	00:20:27	00:42:07
Stop time krabicovače	00:16:14	00:27:20	00:42:31

Tabulka 9.3 - Výsledky simulací 4-6

Na výsledcích je hezky vidět, jak se pomalu zvyšuje doba, kdy jsou stroje v poruše. Na simulaci číslo 5 můžeme pozorovat zvýšení celkové propustnosti linky. V tomto případě by se vyplatilo navýšení rychlostí jednotlivých strojů, protože MTBF je v takovou chvíli ještě únosné a zvýšení rychlosti jej kompenzuje. Při pohledu na simulaci číslo 6 je již patrné, že zde zvýšení rychlosti už přešlo přes jistou mez, kdy se ještě vyplatí. Průměrná rychlost linky klesla velmi značně, a to z důvodů častých zastavení jednotlivých strojů.

Na následujících dvou obrázcích (Obrázek 9.3 a Obrázek 9.4) je možné demonstrovat velký rozdíl mezi tím, kdy je stroj v pořádku a kdy je stroj v poruše.

Při porovnání časů 47 minut a 1 hodina 51 minut, je velice zřejmé, že nemusí být pravidlem, že zvýšení rychlosti strojů znamená vyšší výkon celé linky.



Obrázek 9.3 - Ukázka výstupu ze simulátoru ze simulace 4

Obrázek 9.4 - Ukázka výstupu ze simulátoru ze simulace 6

Kapitola 10

10 Závěr

Analýza výrobní linky ukázala, že Petriho sítě jsou ideální nástroj k jejich simulování. Na základě analýzy se podařilo vytvořit model linky reprezentovaný Petriho sítí a s touto sítí se dá vytvořit neomezeně dlouhá linky s neomezeným počtem dopravníků. Zde se velmi dobře ukázala jednoduchost a robustnost tohoto matematického aparátu. Tím dostává simulátor obrovskou možnost simulovat širokou škálu linek.

Následně se podařilo vytvořit implementaci celé Petriho sítě v jazyce C# a kompatibilního Token playeru implementovaného ve stejném jazyce. Dále byl vytvořen převodník, který je z konfiguračního souboru schopen generovat Petriho síť podle modelu, který vyplynul z analýzy.

Důležité je také, že simulátor je schopen zpracovávat výstup z Token playeru, vypočítat potřebné veličiny a vizualizovat průběh celé simulace.

Simulace 1 - 6 ukazují, že nástroj vykazuje vlastnosti reálné linky a současně odhaluje jemné nijace, které není možné empirickým pozorováním odhalit.

Problematika výrobních linek je a bude stále aktuální téma, protože konzumní společnost bude výrobky z těchto linek požadovat. Není důležité, co linka konkrétně vyrábí, jestli je to nábytek ze dřeva, boty z gumy, plnění produktu do lahviček, nebo odlévání železných nosníků. Důležité je, že všechny tyto linky mají mnohé společné. Tím je, že linka se skládá ze strojů a dopravníků, každý z těchto zařízení má svůj stavový automat a svoje parametry. Na základě těchto znalostní může simulátor představovat velmi robustní nástroj pro optimalizaci chodu linky.

Zdroje

[1] TORELL, Wendy a Victor AVELAR. Mean Time Between Failure: Explanation and Standards. In: [online]. 2004 [cit. 2013-05-23]. Dostupné z: http://www.criticalpowerandcooling.com/white-papers/Architecture/WP-78%20Mean%20Time%20Between%20Failure%20-%20Explanation%20and%20Standards.pdf

[2] Mean Time Between Failures definition. *Dictionary.com* [online]. 1998 [cit. 2013-05-23]. Dostupné z: http://dictionary.reference.com/browse/mean+time+between+failures

[3] PETRI, Carl Adam. *Kommunikation mit automaten*. Bonn, 1962. Dostupné z: http://edoc.sub.uni-hamburg.de/informatik/volltexte/2011/160/pdf/diss_petri.pdf. Desertační práce. Technischen Hochschule Darmstad.

[4] HANZÁLEK, Zdeněk. *Petriho sítě*. Praha, 2008. Dostupné z: http://support.dce.felk.cvut.cz/pub/hanzalek/prednasky/petri_text.pdf. Text k přednáškám. České Vysoké Učení Technické.

[5] MURATA, Tadeo. *Petri Nets: Properties, Analysis and Applications*. Proceeding of the IEEE, vol. 77, No. 4, 1989, s. 541-580.

[6] AJMONE MARSAN, Marco. *Modelling with generalized stochastic petri nets*. Vyd. 1. Boston: J. Wiley, 1995, 301 s. ISBN 04-719-3059-8.

[7] HANZÁLEK,. Zdeněk a SVÁDOVÁ. *Matlab toolbox for petri nets*. Praha. Dostupné z: http://support.dce.felk.cvut.cz/pub/hanzalek/publications/Hanzalek01a.pdf. Dokumentace k Toolboxu. České Vysoké Učení Technické.

Příloha číslo 1

V této příloze je tabulka s veličinami, které musí být uvedeny v konfiguračním souboru.

Úvodní proměnné + poznámky

Jméno proměnné	Význam proměnné
// text	Poznámka
BottleDiametter	Velikost lahve (cm)

Zadání nového stroje je uvozeno slovem „MACHINE“.

Parametry stroje

Jméno proměnné	Význam proměnné
Id	Identifikační číslo stroje
Name	Jméno stroje
NumberOfBottlesInside	Kolik produktů se vejde do stroje (na kolika produktech současně pracuje)
MeanTimeBetweenFailures	Střední doba mezi poruchami (s)
MeanTimeToRepair	Střední doba opravy (s)
SourceAbility	0/1 – schopnost produkovat suroviny nebo produkty
FrequencyOfProducedBottles	Jak často jsou produkty vytvářeny (ppm) – tento parametry není žádoucí pokud *SourceAbility* je nastavena na 0
SinkAbility	0/1 – Schopnost konzumovat produkty
RatioOfConsumedBottles	Jak často jsou produkty konzumovány (ppm) – tento parametr není žádoucí pokud *SinkAbility* je nastaveno na 0
NumberOfStates	Počet stavů, které stroj má. Musí být následováno patřičným počtem stavů. Stav *breakDown* se nezahrnuje do tohoto čísla.
NumberOfTransitions	Počet přechodů mezi jednotlivými stavy. Musí být následováno patřičným počtem přechodů.

Zadání nového deterministického stavu je uvozeno slovem „STATEDET“.

Parametry stavu

Jméno proměnné	Význam proměnné
SpeedNominal	Nominální rychlost (ppm)
Type	Typ stavu – nabývá hodnot z ***MachineStateType.cs*** nebo ***ConveyorStateType.cs***

Zadání nového stavu, který není deterministický, je uvozeno slovem „STATENODET“.

Parametry stavu

Jméno proměnné	Význam proměnné
SpeedNominal	Nominální rychlost (ppm – pro stroj, metry za minutu – pro dopravník)
Type	Typ stavu – nabývá hodnot z ***MachineStateType.cs*** nebo

	ConveyorStateType.cs
MaxSpeedCoeficient	Koeficient, po jehož vynásobení nominální rychlostí dostaneme maximální rychlost v daném stavu.
MinSpeedCoeficient	Koeficient, po jehož vynásobení nominální rychlostí dostaneme minimální rychlost v daném stavu.

Zadání nového dopravníku je uvozeno slovem „CONVEYOR“.

Parametry dopravníku

Jméno proměnné	Význam proměnné
Id	Identifikační číslo dopravníku
Name	Jméno dopravníku
Length	Délka dopravníku (cm)
NumberOfStates	Počet stavů, které dopravník má. Musí být následováno patřičným počtem stavů. Stav *stop* se nezahrnuje to tohoto čísla.
NumberOfTransitions	Počet přechodů mezi jednotlivými stavy. Musí být následováno patřičným počtem přechodů.

Zadání nového přechodu je uvozeno slovem „TRANSITION“.

Parametry přechodu

Jméno proměnné	Význam proměnné
TypeFrom	Název stavu, ze kterého přechod vychází
TypeTo	Název stavu, ve kterém přechod končí
TransitionTime	Doba přecházení ze stavu From do stavu To (s)
TransitionConstatn	Doba, jak dlouho má v daném stavu stroj zůstat. Tento parametr je bezpředmětný, pokud podmínkou přechodu ze stavu From není čas.
SpeedMultiplier	Konstanta, kterou když vynásobíme rychlostí rychlejšího ze stavu, dostaneme rychlost pro přechod.