Kapitola 7

Systém DSI pro diskrétní simulaci

7.1 Systém DSI povšechně

Systém DSI je relativně jednoduchý programový prostředek pro simulaci poměrně široké třídy diskrétních náhodných procesů zejména (ale nejen) z oblasti teorie hromadné obsluhy (teorie front).

Lze jej použít k simulaci otevřených i uzavřených systémů hromadné obsluhy; kanály obsluhy mohou být zapojeny sériově, paralelně i smíšeně; doba obsluhy a pohyb zákazníků systémem může záviset na atributech zákazníků, atributy se mohou při průchodu systémem měnit. Lze však modelovat i řadu dalších procesů, např. z oblasti teorie zásob.

Zkušenosti ukazují, že systém DSI je vhodný nejen pro samotné řešení školních problémů. Pro svou jednoduchost a zároveň přiměřenou sílu je jazyk systému DSI výborným prostředkem pro procvičování nesnadného „umění“ udělat dobrý model.

Systém DSI lze volně používat pouze pro vzdělávací účely na Stavební fakultě ČVUT. Jakékoli jiné použití (na jiných školách, popř. pro komerční účely) je podmíněno písemným souhlasem autora. Autorem systému DSI je RNDr Jiří Demel, CSc, katedra inženýrské informatiky, Stavební fakulta ČVUT, Thákurova 7, 166 29 Praha 6, e-mail `demel@fsv.cvut.cz`.

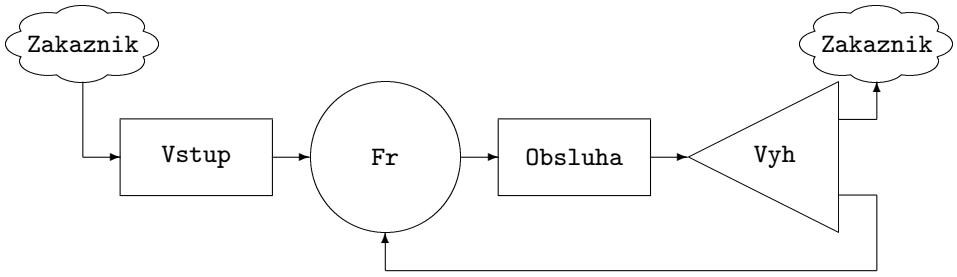
7.1.1 Použití systému DSI. Uživatel vytváří popis simulovaného systému jako ASCII-soubor ve speciálním jazyce DSI. Tento popis je pak sytémem DSI přeložen, čímž vzniká samostatně spustitelný simulační program, využívající metodu proměnného časového kroku (viz 6.3, str. 52). Při běhu tohoto programu se průběh a výsledky simulace zapisují do protokolu. Průběh simulace a podrobnost protokolu lze interaktivně řídit.

Uživatelský interface systému DSI je inspirován Turbo Pascalem (a řadou podobně ovládaných programů) — uživatel vytváří a edituje popis modelu a popis počátečních podmínek pomocí speciálního editoru. Speciálním povelem (stiskem klávesy F9) se přímo z editoru vyvolá překlad a posléze, byl-li překlad úspěšný, se spustí vytvořený simulační program.

Po dokončení simulace se v témže editoru automaticky zobrazí protokol o simulaci. Přitom v dalších oknech editoru lze znovu editovat popis systému i popis počátečních podmínek a znovu klávesou F9 vyvolat překlad a simulaci.

Překlad z jazyka DSI do spustitelné formy probíhá ve dvou fázích: nejprve je popis modelu přeložen do objektové verse jazyka Turbo Pascal, druhou fázi provádí překladač Turbo Pascalu. Obě fáze překladu jsou před uživatelem skryty, systém DSI je provádí automaticky.

Pokud je při překladu nalezena chyba, systém DSI oznámí popis chyby a umístí kursor na místo, kde byla chyba nalezena.



Obrázek 7.1: Ilustrace k příkladu 7.1.3

7.1.2 Simulační objekty. Simulační model DSI si lze představit jako orientovaný graf, jehož vrcholy představují tzv. *simulační objekty* a hrany představují směr pohybu tzv. *entit* mezi simulačními objekty. Než přistoupíme k podrobnému popisu, uvedeme velmi stručné a ne zcela přesné charakteristiky jednotlivých typů simulačních objektů a malý příklad.

Entity mohou představovat v podstatě jakékoli prvky, které se v systému pohybují mezi simulačními objekty. Příkladem mohou být zákazníci nebo prázdné košíky v samoobsluze, dopravní prostředky, výrobky, zboží na skladě a pod.

Fronta je v podstatě skladištěm entit určitého typu. Entity zde pasivně čekají, až budou z fronty vyjmuty některou *aktivitou*.

Aktivita funguje tak, že ze svých vstupních front odebere předepsané počty entit, tyto entity pak po nějakou dobu „zpracovává“ a pak je odešle do svých výstupních front. Podmínkou spuštění (nastartování) aktivity je dostatek entit ve vstupních frontách.

Vyhýbka umožňuje větvit pohyb entit podle nějakých podmínek, například i náhodně.

Pool (anglicky bazén, nádrž, čti půl) je nevyčerpatelný zásobník entit. Chová se jako fronta, ve které je nekonečně mnoho entit. Pro každý typ entity existuje v systému vždy přesně jeden samostatný pool (ale na obrázcích jej kvůli přehlednosti někdy kreslíme několikrát).

7.1.3 Příklad. Uvedeme jednoduchý příklad jednokanálového systému hromadné obsluhy. Od klasického učebnicového modelu M/M/1 se liší jednak dobou obsluhy, jednak tím, že polovině zákazníků se obsluha zalíbila natolik, že si jdou ihned znovu stoupnout do fronty a nechají se znovu obsloužit. (A nebo se obsluha nepodařila a je nutno ji opakovat – z hlediska modelu to může být lhostejné.)

```
entity  Zakaznik;                { Tím je popsán typ entity }

activity  Vstup  channels 1       { aktivita Vstup modeluje  }
  load 1 Zakaznik from pool       { Poissonův vstupní proces }
  after  negexp(0.66667) ::  eject Zakaznik to Fr;
  end;

queue  Fr  of  Zakaznik  FIFO;     { Fronta }

activity  Obsluha  channels 1      { jednokanálová obsluha }
  load  1 Zakaznik from Fr
  after  uniform(0.2, 0.3) ::      { náhodná doba obsluhy }
    eject Zakaznik to Vyh;
  end;

switch  Vyh  for Zakaznik          { Vyhýbka: }
```

```
  case  Random < 0.5 ::  pool,    { 50% zákazníků odchází pryč, }
  otherwise              Fr;      { 50% jde znovu do Fronty. }

end.
```

Systém je graficky znázorněn na obr. 7.1.

V tomto systému pracujeme s entitami jediného typu `Zakaznik`.

Aktivita `Vstup` funguje jako zdroj zákazníků: vždy vezme jednoho zákazníka z poolu, zdrží jej po náhodnou dobu a pak jej odešle do fronty `Fr`. Tím se uvolní (jediný) kanál obsluhy a celý proces se opakuje: opět je odebrán zákazník z poolu, je zdržen a odeslán. Doba zdržení entity v aktivitě tedy určuje délku intervalů mezi odchody zákazníků. V našem případě je tato doba náhodná s exponenciálním rozložením se střední hodnotou $0.66667 \doteq 2/3$, proces „vzniku zákazníků“ je tedy Poissonův proces s intensitou $\lambda = 1.5$ zákazníka za jednotku času.

Fronta `Fr` shromažďuje zákazníky, kteří čekají na obsluhu.

Aktivita `Obsluha` má jeden kanál. Doba obsluhy je náhodná s rovnoměrným rozložením 0.2–0.3 časové jednotky. Obsloužení zákazníci jsou odesíláni do vyhýbky `Vyh`.

Vyhýbka `Vyh` rozděluje zákazníky na základě volání generátoru náhodných čísel. S pravděpodobností 1/2 je zákazník odeslán znovu do fronty `Fr` (a pak znovu obsluhován), ostatní zákazníci jsou odesláni do poolu, tedy vlastně odcházejí ze systému.

7.2 Jazyk DSI

7.2.1 Syntax jazyka DSI. Popis systému v jazyce DSI se skládá z popisů jednotlivých simulačních objektů (klausule `entity`, `queue`, `switch` a `activity`) a případně z nepovinných úseků v jazyce Pascal (klausule `declarations`, `on start` a `on write`.

Přehled syntaxe je uveden v tabulce 7.1, str. 58. V tomto přehledu svislá čára vyjadřuje alternativu, hranaté závorky $[\ldots]$ uzavírají volitelný text (lze jej vynechat), hvězdička $[\ldots]^*$ znamená možnost opakování obsahu předchozí závorky libovolný počet-krát (včetně nulového počtu opakování) a znaménko plus $[\ldots]^+$ znamená opakování alespoň jedenkrát.

Pořadí popisu jednotlivých simulačních objektů je libovolné. Formát popisu je volný: oddělovače řádek nebo řada mezer za sebou mají z hlediska jazyka DSI stejný význam, jako jedna mezera. Text popisu tedy lze po formální stránce přizpůsobit tomu, co uživatel pokládá za přehledné.

Popis sytému může obsahovat komentáře uzavřené do složených závorek. Komentáře lze do sebe vnořovat. Tedy například “`{tato {malá} ukázka}`” je celá komentářem.

Jména (simulačních objektů, atributů) mohou mít délku maximálně 20 znaků. Jméno musí začínat písmenem a smí obsahovat pouze písmena a číslice. Jako jména se nesmí použít klíčová slova a jména funkcí systému DSI, tj.

```
entity, attrib, int, real, bool, str, pool,
queue, of, signal, to, fifo, lifo, dummy, random, prior,
switch, for, case, otherwise,
activity, channels, load, from, compute, after, eject,
declarations, on, start, write, end,
uniform, negexp, normal, intUniform, time,
contents, busyChannels, hasFreeChannel
```

```
⟨popis entity⟩ ::=
      entity ⟨jm entity⟩
      [attrib
        [⟨jm atributu⟩ [, ⟨jm atributu⟩]* : ⟨typ atributu⟩ ; ]*
       end] ;

⟨typ atributu⟩ ::=
        int  [: ⟨počet znaků⟩]
      | real [: ⟨počet znaků⟩ : [⟨desetinných míst⟩]]
      | bool [: ⟨počet znaků⟩]
      | str  [: ⟨počet znaků⟩]

⟨popis fronty⟩  ::=
      queue ⟨jm fronty⟩ of ⟨jm entity⟩  ⟨typ fronty⟩
          [signal [to]  ⟨jm aktivity⟩ [, ⟨jm aktivity⟩]*] ;

⟨typ fronty⟩  ::=  fifo | lifo | random | prior ⟨jm atributu⟩

⟨popis vyhýbky⟩  ::=
      switch ⟨jm vyhýbky⟩  for ⟨jm entity⟩
          [ case ⟨výraz typu boolean⟩ :: ⟨kam⟩ , ]*
          otherwise  ⟨kam⟩;

⟨popis aktivity⟩  ::=
      activity ⟨jm aktivity⟩  channels ⟨počet kanálů⟩
          load
              [ ⟨počet entit⟩ from ⟨odkud⟩, ]+
          [compute ⟨příkaz jazyka Pascal⟩ ::]
          [after ⟨výraz typu real⟩ :: eject
              ⟨jm entity⟩ to ⟨kam⟩
              [, ⟨jm entity⟩ to ⟨kam⟩]*
          ;]+
          end;

⟨odkud⟩  ::=  ⟨jm fronty⟩ | pool
⟨kam⟩  ::=  ⟨jm fronty⟩ | pool | ⟨jm vyhýbky⟩

⟨úsek v jazyce Pascal⟩  ::=
        declarations ⟨deklarace v jazyce Pascal⟩ ::
      | on start ⟨příkaz jazyka Pascal⟩ ::
      | on write ⟨příkaz jazyka Pascal⟩ ::
```

Tabulka 7.1: Přehled syntaxe jazyka DSI.

a dále se nesmí jako jména použít klíčová slova jazyka Pascal. Ve jménech a v klíčových slovech se nerozlišují velká a malá písmena.

Na některých místech popisu systému je nutno použít kousek programu (výraz, příkaz nebo deklaraci) v jazyce Pascal. Každý takový úsek programu je nutno v jazyce DSI ukončit dvěma dvojtečkami těsně za sebou. Překladač jazyka DSI podle nich rozeznává konec Pascalského úseku. Z toho vyplývá (nepodstatné) omezení: pascalský úsek nemůže obsahovat dvě dvojtečky těsně za sebou a to ani v komentáři nebo v textovém řetězci (mezi apostrofy).

Pascalský úsek může obsahovat komentáře; tyto komentáře se však řídí pravidly jazyka Pascal, tj. např. nelze je do sebe vnořovat.

Klausule `declarations` se může v popisu systému vyskytovat několikrát a to vždy na nejvyšší syntaktické úrovni (tj. před, za nebo mezi simulačními objekty). Texty jednotlivých deklarací budou zřetězeny za sebou a budou umístěny do simulačního programu jako součást deklarací.

Klausule `on start` a `on write` se mohou vyskytovat na stejných místech jako `deklarations`, ale každá z nich se smí v popisu sytému vyskytnout nejvýše jedenkrát.

7.2.2 Entita. Entity jsou nositeli veškerého dění v modelovaném systému. Jinak řečeno, cokoli se v systému děje, děje se vlastně s entitami: Většina dějů v systému spočívá v přemístění entity z jednoho simulačního objektu do druhého. Někdy se přitom navíc mění i atributy entity.

Každá entita je nějakého *typu*, přičemž typ entity musí být v jazyce DSI popsán (tj. alespoň pojmenován), pomocí klausule `entity`.

Obecně může existovat libovolný počet exemplářů entit stejného typu. Pro každý typ entity je v systému automaticky vytvořen speciální simulační objekt, tzv. *pool*, který funguje jako potenciálně nekonečná zásoba entit daného typu. Přemístění entity z poolu do některého jiného prvku systému chápeme jako *vznik* entity. Naopak, přemístění entity do poolu chápeme jako její *zánik* (ukončení života) entity. Každá entita je během svého života neustále obsažena v některém simulačním objektu. Přechod entity z jednoho objektu do druhého se vždy děje skokem, tj. má nulové trvání.

Entity mohou (ale nemusí) mít atributy. Popis entity bez atributů má jednoduchý tvar:

⟨*popis entity*⟩ ::= `entity` ⟨*jm entity*⟩`;`

Má-li mít entita atributy, je třeba v definici popsat jména a typy atributů pomocí klausule `attrib`. Atribut může být jednoho ze čtyř typů:

`int`	integer
`real`	real
`bool`	boolean
`str`	řetězec o délce max. 20 znaků

Popis entity s atributy má tvar:

⟨*popis entity*⟩ ::=
`entity` ⟨*jm entity*⟩ `attrib`
[⟨*jm atributu*⟩ [`,` ⟨*jm atributu*⟩]* `:` ⟨*typ atributu*⟩ `;`]*
`end;`

⟨*typ atributu*⟩ ::=
`int` [`:` ⟨*počet znaků*⟩]
| `real` [`:` ⟨*počet znaků*⟩ `:` [⟨*desetinných míst*⟩]]
| `bool` [`:` ⟨*počet znaků*⟩]
| `str` [`:` ⟨*počet znaků*⟩]

Údaj „:⟨*znaků*⟩" určuje, zda v protokolu o simulaci mají být uváděny hodnoty tohoto atributu a kolik znaků má tento údaj zaujímat. Pokud za typem atributu není dvojtečka a číslo, pak to znamená, že v protokolu o simulaci nebude hodnota tohoto atributu uváděna.

Příklady popisů entit:

```
entity  Zakaznik;

entity  Prodavac
  attrib  jmeno   : str:20;
          cislo   : int;
          unava   : real:7:2;
          obedval : bool;
          snidal, svacil  : bool : 5
  end;
```

Zde entita typu `Zakaznik` nemá žádné atributy, mezi různými entitami tohoto typu tedy nemáme možnost rozlišovat (dost často to není potřeba). Entita typu `Prodavac` má šest atributů. Čtyři z nich (`jmeno`, `unava`, `snidal` a `svacil`) budou v protokolu o simulaci uváděny u každé entity typu `Prodavac`. Atribut `unava` bude uváděn na dvě desetinná místa a bude mít šířku 7 znaků včetně znaménka a desetinné tečky. Atributy `cislo` a `obedval` nebudou v protokolu uváděny.

7.2.3 Fronta. Fronta (anglicky queue, čti kjů) slouží k uchovávání entit. Každá fronta má jméno a dále má pevně stanovený typ entit, které může uchovávat.

Entita může do fronty vstoupit buď z objektu typu aktivita nebo typu vyhýbka. V obou případech je fronta povinna přicházející entitu přijmout a uschovat. Počet entit ve frontě není omezen.[1]

Entity mohou z fronty odejít jediným způsobem: mohou být odebrány některým objektem typu aktivita. Pořadí odebíraných entit je určeno tzv. *režimem* fronty, který může být

režim	z fronty se vybírá entita:
`FIFO`	která přišla první (first in first out)
`LIFO`	která přišla poslední (last in first out)
`random`	náhodně s rovnoměrným rozložením
`prior`	podle priority – vybere se entita s nejmenší hodnotou daného atributu

Fronta vždy při příchodu entity signalisuje všem následujícím aktivitám, že se stav fronty změnil a že se tedy mohou pokusit entitu z fronty odebrat. Fronta tuto signalisaci dělá automaticky, uživatel se o ni nemusí starat, může ale v popisu fronty předepsat pořadí signalisace. Toto pořadí totiž může mít podstatný vliv na to, která aktivita nově přišlou entitu odebere.

Pokud v popisu fronty chybí klausule `signal to`, provádí se signalisace v náhodném pořadí. Je-li klausule `signal to` uvedena, pak musí obsahovat jména *všech* aktivit, které z této fronty mohou odebírat entity.

Popis fronty má tvar:

⟨*popis fronty*⟩ ::=
`queue` ⟨*jm fronty*⟩ `of` ⟨*jm entity*⟩ ⟨*režim fronty*⟩
[`signal` [`to`] ⟨*jm aktivity*⟩ [`,` ⟨*jm aktivity*⟩]*] `;`

⟨*režim fronty*⟩ ::= `fifo` | `lifo` | `random` | `prior` ⟨*jm atributu*⟩

[1] Přesněji, počet entit není omezen jazykem DSI, je však během simulace omezen velikostí paměti počítače.

Příklady popisu fronty:

```
queue  qZak  of  Zakaznik  FIFO  signal to  Prodej;

queue  unaveniProdavaci  of  Prodavac  prior unava
    signal to Prodej, PripravaZbozi;
```

Zde fronta `qZak` je klasická „spravedlivá“ fronta, z níž jsou zákazníci odebíráni v pořadí, ve kterém přišli. Klausule `signal to` je v tomto konkrétním případě zbytečná a mohla by být vynechána (proč?).

Fronta `unaveniProdavaci` je prioritní: z fronty se bude vybírat vždy prodavač, který bude mít nejnižší hodnotu atributu `unava`.

Klausule `signal to` zde určuje, že příchod prodavače do fronty bude oznámen aktivitám `Prodej` a `PripravaZbozi` v tomto pořadí. To znamená, že pokud obě tyto aktivity čekaly pouze na příchod prodavače, pak bude spuštěna aktivita `Prodej`, neboť se dozví o příchodu volného prodavače dříve. Kdybychom zde klausuli `signal to` vynechali, pak by, za jinak stejných podmínek, byla náhodně (s pravděpodobností 1/2) spuštěna jedna nebo druhá z obou aktivit.

7.2.4 Aktivita. Aktivita je v systému DSI jediným skutečně aktivním typem simulačních objektů.

Velmi stručně řečeno, aktivita je schopna vzít několik entit ze vstupních front nebo poolů, po nějakou dobu je uchovávat a po uplynutí této doby je odeslat do dalších (pasivních) simulačních objektů. Popis aktivity má tvar:

⟨*popis aktivity*⟩ ::=
`activity` ⟨*jm aktivity*⟩ `channels` ⟨*počet kanálů*⟩
`load`
[⟨*počet entit*⟩ `from` ⟨*odkud*⟩`,`]$^+$
[`compute` ⟨*příkaz jazyka Pascal*⟩ `::`]
[`after` ⟨*výraz typu real*⟩ `::` `eject`
⟨*jm entity*⟩ `to` ⟨*kam*⟩
[`,` ⟨*jm entity*⟩ `to` ⟨*kam*⟩]*
`;`]$^+$
`end;`

⟨*odkud*⟩ ::= ⟨*jm fronty*⟩ | `pool`
⟨*kam*⟩ ::= ⟨*jm fronty*⟩ | `pool` | ⟨*jm vyhýbky*⟩

Aktivita může v systému pracovat v několika kopiích, tzv. *kanálech* (anglicky channel), které mohou pracovat souběžně a nezávisle na sobě. Maximální počet kanálů je určen klausulí `channels` n, kde n je kladné číslo. Je-li uvedena nula, znamená to, že počet kanálů je neomezený.

Klausule `load` (anglicky naložit) určuje, kolik entit, jakého typu a odkud musí aktivita vzít, aby mohla zahájit práci některého svého kanálu.

Jsou-li ve vstupních frontách potřebné počty entit a má-li aktivita volný kanál, pak některý kanál (mezi kanály nerozlišujeme) zahájí svou činnost a to tím, že odebere ze vstupních front a poolů potřebné počty entit. Nejsou-li tyto podmínky splněny (tj. není-li volný kanál nebo mají-li vstupní fronty málo entit), aktivita čeká na jejich splnění. Další pokus o spuštění kanálu se tedy bude činit až po signálu od některé fronty, že do ní přišla entita, nebo po signálu, že byl uvolněn nějaký kanál v téže aktivitě.

Je-li v popisu aktivity uvedena klausule `compute`, provede se ihned (tj. v rámci zahajování práce kanálu) předepsaný výpočet, tj. příkaz jazyka Pascal (může to samozřejmě být i příkaz

složený). Tento příkaz může např. měnit atributy entit, které jsou v kanále obsaženy. Poznamenejme, že zápis příkazu musí být vždy ukončen dvěma dvojtečkami.

Klausule `after` (anglicky po) obsahuje výraz, kterým se vypočte doba zdržení entit v aktivitě. Po uplynutí této doby budou entity odeslány do objektů uvedených v klausuli `eject` (anglicky vypudit). Skupinu entit, které jsou takto vypuzeny současně, nazýváme *dávkou*.

V popisu aktivity může být několik klausulí `after`. Každá z nich určuje jednu dávku. Odesláním poslední dávky (tj. po odeslání všech entit) činnost kanálu končí a kanál se stává volným.

Příklad popisu aktivity:

```
activity  PlaceniUPokladny  channels 3
  load  1  Prodavac from unaveniProdavaci,
        1  Zakaznik from qZak,
        1  kosik    from pool,
  compute  Prodavac^.unava := Prodavac^.unava + 1; ::
  after  Prodavac^.unava+5 :: eject prodavac to unaveniProdavaci,
                                  zakaznik to pool;
  after  Prodavac^.unava+4 :: eject kosik    to qVolneKosiky;
  end;
```

Tato aktivita modeluje placení u pokladny v samoobsluze. Aktivita má tři kanály. To znamená, že paralelně mohou až tři prodavači obsluhovat tři zákazníky. V praxi by počet kanálů byl omezen např. počtem pokladen.

K zahájení činnosti kanálu je třeba (samozřejmě kromě volného kanálu), aby vstupní fronty `unaveniProdavaci` a `qZak` obsahovaly každá alespoň jednu entitu. Po zahájení práce kanálu, tj. po odebrání entit ze vstupních front a košíku z poolu, se (v klausuli `compute`) vypočte (zde zvýší) únava prodavače a potom se vypočtou doby zdržení entit, tj. časy, kdy budou odesílány jednotlivé dávky entit z kanálu pryč. Zde oba časy (tj. doby zdržení entit) závisí na právě vypočtené hodnotě únavy. Všimněte si, že pořadí, v jakém jsou napsány klausule `after`, nemá nic společného s pořadím, ve kterém budou dávky skutečně odesílány. Entita `prodavac` je odeslána do své původní fronty, entita `zakaznik` je odeslána do poolu, tj. odchází ze systému pryč a entita `kosik` je odeslána do fronty volných košíků.

Poznamenejme, že tento model je založen na předpokladu, že uvnitř samoobsluhy se zákazník nesmí pohybovat bez košíku a tedy obsloužením zákazníka u pokladny vlastně jakoby vzniká volný košík. Dále poznamenejme, že klausulí `load 1 kosik from pool` přijímáme v našem příkladě poněkud nerealistický předpoklad, že v samoobsluze je neomezené množství košíků. Realističtější model bychom získali použitím fronty pro volné košíky. Promyslete detaily.

7.2.5 Vyhýbka. Vyhýbka umožňuje větvit pohyb entity v závislosti na splnění nějakých podmínek.

Entity mohou do vyhýbky přicházet z aktivity nebo z jiné vyhýbky a odeslány mohou být do fronty, poolu nebo do další vyhýbky. Průchod entity vyhýbkou má (z hlediska modelového času) nulové trvání. Popis vyhýbky má tento tvar:

⟨*popis vyhýbky*⟩ ::=
`switch` ⟨*jm vyhýbky*⟩ `for` ⟨*jm entity*⟩
[`case` ⟨*výraz typu boolean*⟩ `::` ⟨*kam*⟩ `,`]*
`otherwise` ⟨*kam*⟩`;`

⟨*kam*⟩ ::= ⟨*jm fronty*⟩ | `pool` | ⟨*jm vyhýbky*⟩

Podmínky (tj. výrazy typu boolean) se při příchodu entity vyhodnocují v tom pořadí, v jakém jsou napsány, dokud se nenajde podmínka, která je splněna — entita je pak odeslána do příslušného objektu. Není-li splněna žádná podmínka, entita je odeslána do objektu, určeného klausulí `otherwise`. Poznamenejme, že klausule `otherwise` je v popisu vyhýbky povinná.

Podmínka, podle níž se pohyb entit větví, zpravidla závisí na nějakém atributu procházející entity, může však také záviset na náhodě (přesněji, na hodnotě získané z generátoru náhodných čísel), na modelovém čase, na počtu prvků v nějaké frontě, popř. na výsledku funkční procedury, kterou uživatel definoval v rámci klausule `declarations`.

Příklad popisu vyhýbky:

```
switch  jeUnaven  for  Prodavac
  case  unava > 9  :: qDovolena;
  case  unava > 5  :: qOdpocinek;
  otherwise           unaveniProdavaci;
```

Zde prodavači, jejichž únava přesahuje hodnotu 9, odcházejí na dovolenou, prodavači, jejichž únava je v rozmezí 6–9, si jdou odpočinout a ostatní odcházejí do fronty `unaveniProdavaci`.

7.2.6 Zdroj entit. Často je třeba modelovat opakovaný vznik nějakého prvku, popřípadě opakující se vstup nějakého prvku do systému z jeho okolí.

V systému DSI pro tento účel není žádný specialisovaný objekt typu zdroj, neboť ke generování entit lze velmi jednoduše použít aktivitu. Typický zdroj je popsán takto:

```
activity  jmZdroje  channels 1
  load  1  jmEntity from pool
  compute
     příkazy definující hodnoty atributů entity
    ::
  after   interval ::
    eject jmEntity to  jmFronty
  end;
```

Aktivita, která funguje jeko zdroj entit, má obvykle jeden kanál. Tím je zajištěno, že časový interval, uvedený v klausuli `after`, bude roven časovému intervalu mezi odesláním dvou po sobě jdoucích entit. Je-li výraz *interval* konstantní, pak entity odcházejí ze zdroje zcela pravidelně (a deterministicky).

Známý Poissonův proces s intenzitou λ [entit za jednotku času], který se často vyskytuje v teorii front, lze modelovat tak, že jako *interval* použijeme výraz `negexp (`λ`)`.

Klausuli `compute` lze vynechat, není-li třeba definovat hodnoty atributů.

Podobným způsobem lze generovat entity i ve skupinách.

7.2.7 Standardní pascalské funkce systému DSI. V sytému DSI lze použít pět předdefinovaných funkcí pro generování pseudonáhodných čísel:

```
function  Random : real;
function  Uniform (A, B : real) : real;
function  NegExp (strHodnota : real) : real;
function  Normal (strHodnota, smerOdchylka : real) : real;
function  intUniform (i, j : integer) : integer;
```

Tyto funkce generují tato rozložení:

`Random`	rovnoměrné v intervalu $\langle 0, 1\rangle$
`Uniform`	rovnoměrné v intervalu $\langle A, B\rangle$
`NegExp`	exponenciální
`Normal`	normální
`intUniform`	celočíselné rovnoměrné v intervalu $\langle i, j\rangle$

Poznamenejme, že funkce `Random` je totožná se standardní funkcí Turbo Pascalu.

Dále jsou k disposici čtyři funkce, které poskytují informace o stavu simulovaného systému:

```
function  Time : real;
function  contEnts (q : queue) : integer;
function  busyChannels (a : activity) : integer;
function  hasFreeChannel (a : activity) : boolean;
```

Funkce `Time` poskytuje aktuální hodnotu modelového času. Je-li tato funkce použita v aktivitě (v klausuli `compute` nebo `after`), pak vrací hodnotu modelového času v okamžiku zahájení práce kanálu.

Funkce `contEnts` vrací počet entit v dané frontě. Je to užitečné např. ve vyhýbce, chceme-li odeslat zákazníka do nejkratší z několika front.

Podobní použití mají i další dvě funkce. Funkce `busyChannels` vrací počet aktivních kanálů v dané aktivitě. Funkce `hasFreeChanel` vrací logickou hodnotu `true`, má-li daná aktivita alespoň jeden volný kanál.

7.2.8 Práce s atributy entit. Tam, kde je z kontextu jasné, o jaký typ entity se jedná (tj. v popisu vyhýbky nebo fronty), používáme k označení atributu samotné jméno atributu. Ukázkou takového použití atributu je výše uvedený příklad popisu vyhýbky, viz 7.2.5, str. 63.

V aktivitě (přesněji, v jejím kanále) se obecně může vyskytovat současně několik entit se stejně pojmenovanými atributy.

Vyskytuje-li se v kanále entita typu E pouze v jediném exempláři (tj. bylo-li uvedeno `load 1` E), pak použití jejího atributu *atr* má tvar

E`^.`*atr* , tedy např. `prodavac^.unava`

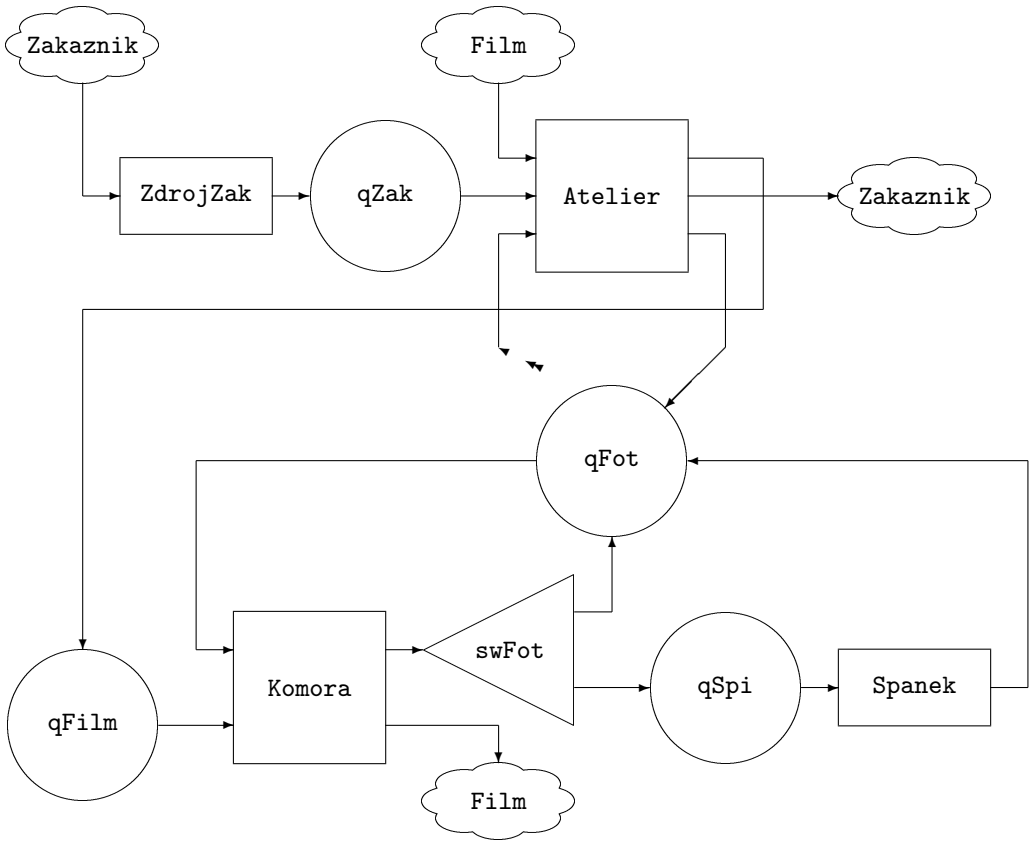
Vyskytuje-li se v kanále entita typu E ve více exemplářích (tj. bylo-li uvedeno `load` n E, kde $n > 1$), pak je nutno dokonce rozlišit mezi jednotlivými exempláři entit téhož typu E. Použití atributu *atr* i-tého exempláře entity má tvar

E`[`i`]^.`*atr* , tedy např. `prodavac[2]^.unava`

7.2.9 Použití klausulí `declarations`, `on start` a `on write`. Tyto klausule značně rozšiřují možnosti systému DSI. Jejich použití však předpokládá trochu vyšší znalost programování.

Klausule `declarations` dovoluje například, aby uživatel definoval své vlastní generátory náhodných čísel, zejména pro náhodné veličiny s empiricky daným rozdělením.

Je také možné definovat globálně přístupná data a tato data pak používat v klausulích `after`, `compute`, popř. `case`. Takovéto použití globálních deklarací je sice možné a legální, jde však do značné míry proti duchu jazyka DSI a lze je doporučit spíše zkušeným programátorům.



Obrázek 7.2: K příkladu 7.2.10 – Fotografický ateliér

Totéž se týká i kalusulí `on start` a `on write`. V těchto klausulích lze uvést příkazy, které se vykonají těsně po spuštění simulačního programu resp. těsně po vykonání příkazu `write`, který zapisuje stav a statistické údaje do protokolu o simulaci. Tyto klausule dovolují např. získat (tj. vypočítat a zapsat do protokolu) podrobnější statistiku o činnosti simulovaného systému, ovšem opět, lze to doporučit spíše zkušenému programátorovi.

Klausule `declarations`, `on start` a `on write` lze zapisovat v souboru `.DSI` kdekoli na nejvyšší syntaktické úrovni, tj. před, za nebo mezi popisy jednotlivých simulačních objektů, nemohou se však vyskytovat uvnitř popisu těchto objektů.

7.2.10 Příklad – fotografický ateliér. Fotograf fotografuje zákazníky, kteří přišli do čekárny. Není-li v čekárně zákazník a má-li fotograf alespoň 10 nevyvolaných snímků, odchází na 10 minut do temné Komory. Po výstupu z Komory, není-li v čekárně žádný zákazník, jde na 15 minut odpočívat. Pak se vrací k obsluze zákazníků. Viz obr. 7.2.

Příchody zákazníků tvoří Poissonův proces s intensitou 5 zákazníků za hodinu, tj. intervaly mezi příchody zákazníků mají exponenciální rozložení se střední hodnotou 12 minut.

```
entity  Zakaznik attrib
  navic : real:5:2   {čas, který zákazník stráví v Ateliéru navíc}
  end;
entity  Fotograf;
entity  film;
```

```
Queue  qZak  of Zakaznik fifo;
qUeue  qFot  of Fotograf fifo  signal to  Atelier, Komora;
quEue  qFilm of Film     fifo;
queUe  qSpi  of fotograf fifo;

switch  swFot for  Fotograf
  case  contEnts (qZak) > 0 :: qFot,
  otherwise                  qSpi;

activity  ZdrojZak  channels 1
  load
    1 Zakaznik from pool,
  compute
    zakaznik^.navic:=Uniform (2, 7);       { Doporučuje se definovat }
    ::                                     { hodnoty všech atributů  }
  after  negExp (12.0) :: eject
    zakaznik to qZak;
  end;

activity  Atelier  channels 1
  load
    1 zakaznik from qZak,
    1 fotograf from qFot,
    1 film     from pool,
  after  uniform (5.0, 6.0) + zakaznik^.navic  :: eject
    zakaznik to pool
    fotograf to qFot
    film     to qFilm;
  end ;

activity  Komora  channels 1
  load
    10 film     from qFilm,
    1  fotograf from qFot,
  after 10 ::  eject
    film     to pool,
    fotograf to swFot;
  end ;

activity  Spanek channels 1
  load
    1 fotograf from qSpi,
  after 15 ::  eject
    fotograf to qFot;
  end ;

end.
```

7.3 Běh simulačního programu

Simulační program ihned po svém spuštění hledá soubor, který má stejné jméno jako on sám a extensi `.ini`. Pokud takový soubor najde, program z něj čte příkazy a každý příkaz ihned vykoná. Pokud soubor `.ini` není nalezen nebo je-li už celý přečten (a vykonán), zobrazí se menu, kterým lze program ovládat interaktivně.

V souboru `.ini` nebo v menu lze použít tyto příkazy:

`randomize ...`	inicialisace generátoru náhodných čísel
`queue ...`	inicialisace front
`till ...`	simulace do daného času
`next ...`	simulace po danou dobu
`step`	jeden krok simulace
`steps ...`	daný počet kroků simulace
`debug on`	zapne ladicí informace v protokolu
`debug off`	vypne ladicí informace v protokolu
`show`	zobrazí stav simulovaného systému
`write`	zapíše stav simulovaného systému do protokolu
`menu`	přepne program do interaktivního režimu
`file`	přepne program do neinteraktivního režimu
`quit`	ukončí práci programu

V souboru `.ini` mohou být příkazy zapisovány bez ohledu na řádky. Jednotlivé příkazy i jejich části se oddělují mezerami.

Podrobný popis je uveden dále. Již nyní však uveďme jednoduchý příklad:

```
debug on   steps 10   debug off
menu
till 1000
write quit
```

Zde prvý řádek obsahuje tři příkazy: prvým příkazem jsou zapnuty ladicí informace, druhý příkaz provede 10 simulačních kroků (do protokolu o simulaci se přitom zapisují podrobné informace o dění v systému) a třetím příkazem jsou ladicí informace vypnuty.

Příkaz na druhém řádku způsobí přechod do interaktivního režimu, v němž uživatel dává příkazy výběrem z menu. V interaktivním režimu lze program i ukončit. Pokud však uživatel zvolí v menu příkaz `file`, program se vrátí ke čtení příkazů ze souboru `.ini` a bude pokračovat příkazem na třetím řádku.

Příkaz na třetím řádku způsobí pokračování v simulaci do modelového času 1000. Dva příkazy na čtvrtém řádku způsobí zápis stavu a statistik do protokolu a ukončení programu.

7.3.1 Protokol o simulaci. Simulační program během své činnosti zapisuje různé informace do protokolu, což je soubor se stejným jménem jako model a s extensí `.out`. Do protokolu se zaznamenávají všechny vykonané příkazy ze souboru `.ini` nebo dané interaktivně. Dále se do něj zapisuje stav systému příkazem `write` a ladicí informace, jsou-li zapnuty příkazem `debug on`.

Poznamenejme, že příkaz `quit` do protokolu nic nezapisuje. Proto, chceme-li z běhu simulačního programu získat statistické údaje, musíme před příkazem `quit` dát příkaz `write`.

Do protokolu může zapisovat i uživatel pomocí příkazu `writeln`. Soubor je deklarován jako `var Fou:text;`. Tedy například klausule

```
compute
  writeln (Fou, 'Hola, hola, přestávka zahájena v čase ', Time:10:3);
  ::
```

způsobí, že při každém spuštění kanálu dané aktivity se do protokolu zapíše daný text. Takto zapsané texty pak lze statisticky vyhodnotit a získat tak údaje, které systém DSI sám běžně neposkytuje.

7.3.2 Inicialisace generátoru náhodných čísel. Simulační program si automaticky inicialisuje generátor náhodných čísel hodnotou 12345.

Uživatel může inicialisaci generátoru změnit pomocí některého z příkazů

```
randomize ⟨celé číslo⟩
randomize time
```

Prvý tvar příkazu způsobí inicialisaci daným číslem.

Druhý tvar způsobí inicialisaci závislou na čase, takže při opakovaných bězích simulačního programu budou použity různé inicialisace, aniž by bylo třeba v inicialisačním souboru cokoli měnit. Hodnota, která byla použita k inicialisaci, je zapsána do protokolu, aby bylo možno i takovýto „náhodně inicialisovaný“ simulační běh reprodukovat.

Příkaz `randomize` lze použít pouze v inicialisačním souboru (tj. nikoli v interaktivním režimu) a pouze před zahájením simulačního běhu.

7.3.3 Inicialisace front. Při startu simulačního progarmu je celý systém prázdný, tj. neobsahuje žádnou entitu. Před zahájením vlastní simulace lze počáteční obsah front definovat pomocí příkazu `queue`. Příkaz `queue` má dva tvary. Jednodušší tvar určuje pouze počet entit, které se mají přidat do fronty:

```
queue ⟨jm fronty⟩ ⟨počet⟩
```

Poznamenejme, že typ entit je pro frontu předem pevně stanoven. Má-li entita atributy, pak jejich hodnoty jsou nedefinované. V řadě případů to nevadí. Pokud ovšem potřebujeme hodnoty atributů předepsat, je třeba použít složitější tvar příkazu:

```
queue ⟨jm fronty⟩
  [entity
    [⟨jm atributu⟩ ⟨hodnota⟩]+
    //
  ]+  end
```

Zde každý výskyt klausule `entity` definuje jeden exemplář entity; popis atributů končí dvojicí lomítek, celý příkaz `queue` pak končí slovem `end`.

Příklad:

```
queue unaveniProdavaci
  entity  svacil t  obedval f  unava 7  //
  entity  svacil f             unava 3  //
  end

queue  uPokladny  2
queue  uPokladny  3
```

Zde prvý příkaz `queue` způsobí, že do fronty `unaveniProdavaci` budou přidány dvě entity. Druhá z těchto entit nemá definovanou hodnotu atributu `obedval`.

Druhá dvojice příkazů přidává entity do téže fronty, celkem tedy bude do fronty `uPokladny` přidáno 5 entit.

Příkaz `queue` se smí vyskytovat pouze v inicialisačním souboru (tj. nelze jej dát v interaktivním režimu) a to před zahájením simulace.

7.3.4 Vlastní simulace. Simulace probíhá v krocích, z nichž každý představuje zpracování jedné (nejbližší) plánované události. Událostí zde je míněno odeslání dávky entit z aktivity.

Uživatel má k disposici čtyři příkazy pro řízení simulace:

```
till ⟨číslo⟩
next ⟨číslo⟩
step
steps ⟨číslo⟩
```

Příkaz `till` provádí simulaci až do dosažení daného modelového času (till znamená až do). Příkaz `next` provádí simulaci po následujících časový interval (modelového času) o dané délce (next znamená následující).

Je třeba zdůraznit, že oba příkazy `till` i `next` se týkají *modelového* času (tj. času, který je modelován) a nikoli doby, jak dlouho bude pracovat simulační program.

Příkaz `step` provede jeden krok simulace, příkaz `steps` provede daný počet kroků. Oba tyto příkazy jsou užitečné zejména při ladění.

Vždy po provedení 100 simulačních kroků program napíše na obrazovku informaci o dosaženém simulačním čase. Běh simulace lze v případě potřeby přerušit stiskem klávesy Esc. Program pak přejde do interaktivního režimu (tj. zobrazí menu) a očekává příkazy od uživatele.

7.3.5 Ladicí informace. Během simulace může program zapisovat do protokolu velmi podrobné informace o tom, co se v simulovaném systému děje. Na začátku (při spuštění simulačního programu) je zápis ladicích informací vypnut. Uživatel může ladicí informace zapínat nebo vypínat pomocí příkazů

```
debug on
debug off
```

Jako příklad uveďme ladicí informace o prvých třech krocích systému z příkladu 7.1.3, str. 56

```
Debug on
Steps 10           (Time=       0.000)
     0.000 :: spuštěn kanál 1  aktivity  Vstup
     0.000 :: entita  Zakaznik  odebrána z POOLu
     0.000 :: dávka 1  plánována na         0.406
     0.000 :: zrušena šance pro  Vstup
     0.000 :: zrušena šance pro  Obsluha

     0.406 :: odeslána dávka 1  kanálu 1  z aktivity  Vstup
     0.406 :: do fronty  Fr  zařazena entita  Zakaznik
     0.406 :: registrována šance pro aktivitu  Obsluha  (pořadí  1)
     0.406 :: ukončen kanál 1  aktivity  Vstup
     0.406 :: registrována šance pro aktivitu  Vstup  (pořadí  2)
     0.406 :: spuštěn kanál 1  aktivity  Obsluha
     0.406 :: z fronty  Fr  odebrána entita  Zakaznik
     0.406 :: dávka 1  plánována na         0.646
     0.406 :: zrušena šance pro  Obsluha
     0.406 :: spuštěn kanál 1  aktivity  Vstup
     0.406 :: entita  Zakaznik  odebrána z POOLu
     0.406 :: dávka 1  plánována na         1.032
     0.406 :: zrušena šance pro  Vstup
```

```
0.646 :: odeslána dávka 1  kanálu 1  z aktivity  Obsluha
0.646 :: vyhýbkou  Vyh  prošla entita  Zakaznik --> Pool
0.646 :: entita  Zakaznik  odeslána do POOLu
0.646 :: ukončen kanál 1  aktivity  Obsluha
0.646 :: registrována šance pro aktivitu  Obsluha  (pořadí  1)
0.646 :: zrušena šance pro  Obsluha

1.032 :: odeslána dávka 1  kanálu 1  z aktivity  Vstup
1.032 :: do fronty  Fr  zařazena entita  Zakaznik
1.032 :: registrována šance pro aktivitu  Obsluha  (pořadí  1)
1.032 :: ukončen kanál 1  aktivity  Vstup
1.032 :: registrována šance pro aktivitu  Vstup  (pořadí  2)
1.032 :: spuštěn kanál 1  aktivity  Obsluha
1.032 :: z fronty  Fr  odebrána entita  Zakaznik
1.032 :: dávka 1  plánována na         1.313
1.032 :: zrušena šance pro  Obsluha
1.032 :: spuštěn kanál 1  aktivity  Vstup
1.032 :: entita  Zakaznik  odebrána z POOLu
1.032 :: dávka 1  plánována na         1.139
1.032 :: zrušena šance pro  Vstup
```

Na začátku řádky (před dvěma dvojtečkami) je vždy uvedena hodnota modelového času.

V této ukázce je uveden i nultý simulační krok, který spočívá v tom, že všechny aktivity (v pořadí, jak byly uvedeny v popisu systému) se pokoušejí využít svou počáteční šanci ke spuštění kanálu. Počáteční šance je udělena vždy všem aktivitám. Zde tuto šanci využila pouze aktivita `Vstup` a to jedenkrát. Další kanál v aktivitě `Vstup` již nebyl k disposici, proto byla šance zrušena. Aktivita `Obsluha` šanci využít nemohla, neboť neměla ve své vstupní frontě žádnou entitu, šance tedy byla také zrušena.

Další simulační kroky vždy začínají zpracováním naplánované události, tj. odesláním dávky entit z některého kanálu některé aktivity. Simulační krok pokračuje zpracováním důsledků této události. Jde vždy o registraci šancí pro nějaké aktivity, jejich případné využití (nastartováním kanálů) a nakonec zrušení.

7.3.6 Informace o stavu simulovaného systému. Výpis stavu systému lze zísakt dvěma příkazy:

```
show
write
```

Příkaz `write` zapíše stav do protokolu, příkaz `show` slouží k prohlížení stavu systému na obrazovce, tj. bez zápisu do protokolu. Formát výpisu stavu je v obou případech stejný.

Výpis stavu simulovaného systému se skládá ze tří částí:

- obsah front a aktivit,
- seznam plánovaných událostí,
- statistické údaje.

Obsah front a aktivit udává pro každou frontu a pro každý aktivní kanál aktivity buď pouze počet obsažených entit a nebo kompletní seznam entit s hodnotami jejich atributů. Který z obou tvarů je použit, záleží na tom, zda v definici atributů entity byl uveden za dvojtečkou počet znaků (viz 7.2.2, str. 60).

Dále uvedené ukázky se opět týkají příkladu 7.1.3, str. 56

```
---------- výpis stavu v čase    100.000
aktivita  Vstup
  kanál    1  obsahuje    1  dávek
    entita  Zakaznik
fronta  Fr  obsahuje  4  entit  Zakaznik
aktivita  Obsluha
  kanál    1  obsahuje    1  dávek
    entita  Zakaznik
---------- konec výpisu stavu v čase    100.000
```

Seznam plánovaných událostí poskytuje přehled, které procesy v simulovaném systému byly aktivní v okamžiku přerušení simulace. Každá řádka seznamu říká, kdy bude odeslána kolikátá dávka entit ze kterého kanálu které aktivity.

Příklad:

```
---------- seznam plánovaných událostí v čase     100.000
na      100.219  plánována dávka 1  kanálu 1  aktivity Obsluha
na      100.496  plánována dávka 1  kanálu 1  aktivity Vstup
---------- konec seznamu plánovaných událostí v čase     100.000
```

Obsah statistických údajů je popsán dále.

7.3.7 Statistické údaje. Statistické údaje jsou nejdůležitější informací, která je výsledkem simulace.

S každým simulačním objektem typu aktivita, fronta nebo pool je v systému DSI spojena vždy jedna celočíselná veličina, která se během simulace mění. Význam této veličiny závisí na typu objektu:

typ objektu	sledovaná veličina
fronta	počet entit ve frontě
pool	počet entit mimo pool (počet „živých“ entit)
aktivita	počet aktivních kanálů

Pro každou z těchto veličin simulační program vypočte tyto údaje:

označení	význam údaje
`#i`	počet, kolikrát veličina vzrostla (i=increase)
`#d`	počet, kolikrát veličina klesla (d=decrease)
`max`	maximum
μ`#`	průměrná hodnota za dobu simulace
μ`T`	průměrný čas mezi vzrůstem a poklesem veličiny

Konkrétně pro jednotlivé typy objektů tyto údaje znamenají:

označení	význam údaje pro **frontu**
`#i`	počet entit, které vstoupily do fronty
`#d`	počet entit, které byly odebrány z fronty
`max`	maximální počet entit ve frontě
μ`#`	průměrný počet entit ve frontě
μ`T`	průměrná doba pobytu entity ve frontě

označení	význam údaje pro **pool**
`#i`	počet entit, které byly odebrány z poolu, tj. přišly do systému
`#d`	počet entit, které byly odslány do poolu, tj. odešly ze systému
`max`	maximální počet entit daného typu v systému
μ`#`	průměrný počet entit daného typu v systému
μ`T`	průměrná doba „života“ entity v systému

označení	význam údaje pro **aktivitu**
`#i`	počet, kolikrát byl nastartován některý kanál aktivity
`#d`	počet, kolikrát byla činnost kanálu ukončena
`max`	maximální počet paralelně pracujících kanálů
μ`#`	průměrný počet paralelně pracujících kanálů
μ`T`	průměrná doba činnosti kanálu

Pokud jsou z aktivity odesílány entity v několika dávkách, pak dobou činnosti kanálu rozumíme dobu od spuštění kanálu do odeslání poslední dávky z tohoto kanálu.

Pro vyhýbky se zaznamenávají a zapisují do protokolu pouze počty entit, které prošly jednotlivými větvemi vyhýbky a celkový počet entit.

Ukázka výpisu statistik se opět týká příkladu 7.1.3, str. 56:

```
---------- výpis statistik v čase    100.000
pool  Zakaznik   152=#i  146=#d    8=max    2.8443=μ#    1.8712=μT
act   Vstup      152=#i  151=#d    1=max    1.0000=μ#    0.6579=μT
que   Fr         281=#i  277=#d    7=max    1.1447=μ#    0.4074=μT
act   Obsluha    277=#i  276=#d    1=max    0.6996=μ#    0.2526=μT
swtch Vyh        276=total#    146 --> Pool
                               130 --> Fr
---------- konec výpisu statistik v čase    100.000
```

7.4 Příklady a cvičení

7.4.1 Příklad – určení počtu opravářů. V dílně, kde pracuje velká skupina strojů je doba mezi poruchami náhodná a má exponenciální rozložení se střední hodnotou 5 minut. Doba opravy jednoho stroje je náhodná s rovnoměrným rozložením 16–18 minut. Ztráty vzniklé prostojem jednoho stroje jsou 500 Kč/hod. Mzda opraváře včetně režie je 100 Kč/hod. Úkolem je zjistit, při jakém počtu opravářů budou ztráty podniku minimální.

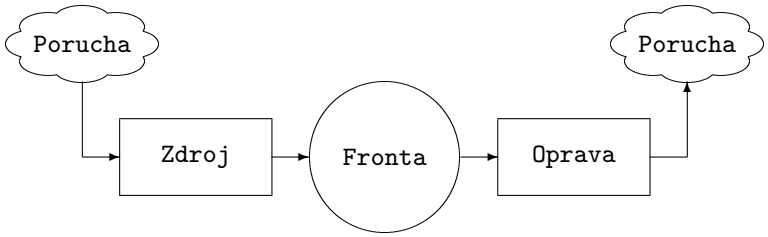
Řešení: Průměrná doba opravy je 17 minut a během této doby vznikne v průměru více než tři poruchy. Opraváři tedy musí být alespoň čtyři. Pro počty opravářů 4, 5, 6, atd., zjistíme (pomocí simulace) průměrné počty strojů mimo provoz.

Příslušný model v jazyce DSI pro čtyři opraváře může vypadat takto:

```
entity  Porucha;

activity  Zdroj  channels 1
  load 1 Porucha from pool
  after  negexp(5) ::  eject Porucha to Fronta;
  end;

queue  Fronta  of  Porucha  FIFO;
```



Obrázek 7.3: Ilustrace k příkladu 7.4.1, str. 72

```
activity  Oprava  channels 4
  load  1 Porucha from Fronta
  after  uniform(16, 18) ::
    eject Porucha to pool;
  end;

end.
```

Systém je graficky znázorněn na obr. 7.3. Pro jiné počty opravářů stačí změnit počet kanálů v aktivitě `Oprava`.

Pro čtyři opraváře a dobu simulace 1440 minut (= třikrát 8 hodin) získáme z protokolu o simulaci tyto údaje:

```
---------- výpis statistik v čase   1440.000
pool  Porucha   253=#i  250=#d  13=max   5.1555=μ#  29.3436=μT
act   Zdroj     253=#i  252=#d   1=max   1.0000=μ#   5.6917=μT
que   Fronta    252=#i  252=#d   8=max   1.1922=μ#   6.8125=μT
act   Oprava    252=#i  250=#d   4=max   2.9633=μ#  16.9333=μT
---------- konec výpisu statistik v čase   1440.000
```

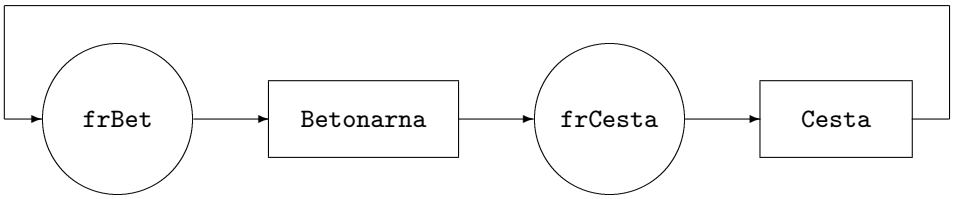
Údaj, který potřebujeme, totiž průměrný počet porouchaných strojů, zde sice přímo uveden není, ale můžeme jej z uvedených údajů velmi snadno vypočítat: Průměrný počet poruch v celém systému je 5.1555. Z toho přesně jedna porucha je neustále v aktivitě `Zdroj`. Ve zbývajících objektech tedy je v průměru celkem 4.1555 porouchaného stroje. Stejný údaj můžeme získat také sečtením průměrných počtů entit ve frontě a v opravě. Pro čtyři opraváře tedy máme průměrnou ztrátu $(4 \cdot 100 + 500 \cdot 4.1555) = 2477.75$ Kč/hod.

Podobně (pouze změnou počtu kanálů) dostaneme

počet opravářů	průmměrný počet strojů mimo provoz	průměrné ztráty Kč
4	4.1555	2477.75
5	3.7718	2385.90
6	3.2214	2210.70
7	3.1621	2281.05
8	3.1859	2392.65

Nejnižší ztráty tedy jsou při 6 opravářích.

7.4.2 Cvičení. Určitou (nepodstatnou) nevýhodou postupu uvedeného v předchozím příkladě je fakt, že pro každý zkoumaný počet opravářů je nutno použít jiný model a z něj odvozený simulační program, přičemž jednotlivé verse se liší pouze počtem kanálů v aktivitě `Oprava`. Pokuste se odstranit tuto nevýhodu. Návod: použijte další entitu typu `Opravar`.



Obrázek 7.4: Ilustrace k příkladu 7.4.4, str. 74 (betonárna)

7.4.3 Cvičení. Řešte stejnou úlohu jako v předchozím příkladě 7.4.1, str. 72 s tím, že se do konkursu na opraváře přihlásili různě výkonní opraváři s různými mzdovými požadavky.

počet opravářů	doba opravy	požadovaná mzda
2	20–24 min	60 Kč
5	16–20 min	80 Kč
3	15–17 min	100 Kč
1	12–16 min	130 Kč

Poznamenejme, že je-li v systému několik opravářů s různým výkonem, je třeba rozhodnout, zda při možnosti volby má být přednostně obsazen opravář nejrychlejší, nejpomalejší, nebo mají-li opraváři být obsazováni náhodně a případně s jakými pravděpodobnostmi. To ovšem není problém tvůrce modelu, to musí rozhodnout provozovatel systému. Tvůrce modelu mu ovšem může v tomto rozhodování pomoci vyčíslením vlivů jednotlivých možných rozhodnutí.

7.4.4 Příklad – betonárna. Z centrální betonárny se rozváží beton pěti automobily se stejnou kapacitou. Výroba jedné dávky trvá přesně 5 minut. Doba potřebná pro dopravu betonu, jeho složení na místě spotřeby a jízdu zpět je náhodná s empiricky zjištěnou distribuční funkcí:

x	$F(x)$
10	0.00
15	0.06
20	0.14
25	0.52
30	0.74
35	0.91
40	1.00

Naším úkolem je zjistit koeficient využití betonárny, tj. procento pracovní doby, kdy je betonárna v činnosti, koeficient využití automobilů, tj. procento pracovní doby kdy je automobil v betonárně nebo na cestě a konečně průměrný počet, kolikrát za hodinu bude betonárna čekat na příjezd automobilu.

Řešení: Pro začátek ponechme stranou požadavek vyčíslit, kolikrát bude betonárna čekat. Model je znázorněn na obr. 7.4, str. 74.

```
entity  Auto;
queue   frBet    of Auto fifo;  { fronta na betonárnu }
queue   frCesta  of Auto fifo;  { fronta na cestu s betonem }
```

```
activity  Betonarna  channels 1
  load 1 Auto from frBet
  after 5 ::  eject Auto to frCesta;
  end;

activity  Cesta  channels 0
  load 1 Auto from frCesta
  after dobaJizdy ::  eject Auto to frBet;
  end;

declarations
  function  dobaJizdy : real;
    type  pole = array [1..7] of real;
    const  F : pole = (0.00, 0.06, 0.14, 0.52, 0.74, 0.91, 1.00);
           X : pole = (10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0);
    var  y : real;  i : integer;
  begin
    y:=random;
    i:=1;
    repeat  i:=i+1  until  F[i] >= y;
    dobaJizdy := x[i] - (F[i]-y) * (x[i]-x[i-1]) / (F[i]-F[i-1]);
  end;
::
end.
```

Frontu `frBet` je třeba inicialisovat příkazem `queue frBet 5`.

Dobu simulace zvolíme dlouhou (100 hodin), abychom potlačili vliv počátečnáho stavu, kdy všechny automobily čekají ve frontě `frBet`. Z protokolu o simulaci získáme tyto údaje:

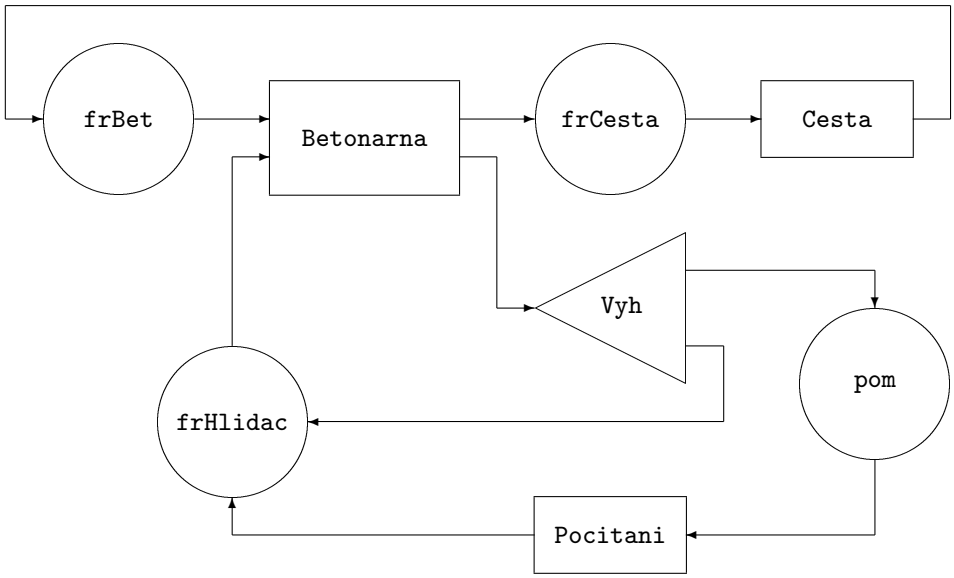
```
---------- seznam plánovaných událostí v čase    6000.000
na     6003.438  plánována dávka 1  kanálu 1  aktivity Betonarna
na     6008.409  plánována dávka 1  kanálu 2  aktivity Cesta
na     6011.367  plánována dávka 1  kanálu 4  aktivity Cesta
na     6013.771  plánována dávka 1  kanálu 1  aktivity Cesta
na     6018.963  plánována dávka 1  kanálu 5  aktivity Cesta
---------- konec seznamu plánovaných událostí v čase    6000.000


---------- výpis statistik v čase   6000.000
pool  Auto          5=#i    0=#d   5=max   5.0000=μ# 6000.0000=μT
que   frBet       916=#i  916=#d   5=max   0.3608=μ#    2.3633=μT
que   frCesta     915=#i  915=#d   1=max   0.0000=μ#    0.0000=μT
act   Betonarna   916=#i  915=#d   1=max   0.7628=μ#    4.9962=μT
act   Cesta       915=#i  911=#d   5=max   3.8764=μ#   25.4193=μT
---------- konec výpisu statistik v čase   6000.000
```

Koeficient využití betonárny získáme přímo jako průměrný počet aktivních kanálů v aktivitě `Betonarna`. Je to přibližně 76 %.

Zjistit využití automobilů je trochu složitější: Sečtením průměrných časů, které stráví auto ve frontách a aktivitách dostaneme průměrnou dobu cyklu 32.7788 minut. Průměrná doba strávená ve frontě `frBet` je 2.3633, což je 7.2 % z doby cyklu. Koeficient využití auta je tedy 92.8 %

Všimněte si průměrné doby činnosti betonárny: na prvý pohled to vypadá jako chyba (třeba zaokrouhlovací) — správně by přece mělo vycházet přesně 5 minut! Vysvětlení je toto: simulace



Obrázek 7.5: Druhá verse modelu k příkladu 7.4.4 (betonárna)

byla přerušena v době, kdy probíhala obsluha. Je to vidět jednak ze statistik (`#i>#d`), jednak ze seznamu plánovaných událostí (v čase 6003.438 je plánováno ukončení aktivity `Betonarna`). Prvých 915 aut bylo obsluhováno vždy plných 5 minut, ale poslední nedokončená obsluha je zde do průměru započítána pouze jako $5 - 3.438 = 1.562$ minuty. Z toho vychází průměrná doba obsluhy 4.9962, takže výše uvedený údaj je správný.

Nyní se zaměříme na zjištění počtu, kolikrát bude betonárna čekat na příjezd auta. Ve statistikách, které systém DSI poskytl pro výše uvedený model potřebné informace nejsou, musíme si tedy pomoci nějakým trikem.

Především si uvědomme, že počet, kolikrát bude betonárna čekat, je roven počtu, kolikrát betonárna ukončí obsluhu a fronta `frBet` bude v tom okamžiku prázdná. Podaří-li se nám zjistit celkový počet těchto situací za celou dobu simulace, můžeme tento počet vydělit dobou simulace v hodinách, čímž získáme požadovaný údaj.

Máme v podstatě dvě možnosti. Prvá možnost spočívá v přidání simulačních objektů a v úpravě modelu tak, aby ze statistik běžně poskytovaných systémem DSI bylo možno odvodit údaje, které potřebujeme. Druhá možnost spočívá ve sběru požadovaných informací prostřednictvím pascalských příkazů vyvolaných v klausulích `compute` nebo při průchodu entity vyhýbkou. Prvý způsob více odpovídá duchu jazyka DSI, druhý je zpravidla jednodušší. Nejprve uvedeme prvý způsob.

Do systému přidáme novou entitu `Hlidac`, která bude vstupovat do aktivity `Betonarna` společně s autem, ale po dokončení obsluhy projhde vyhýbkou, kde větvení bude záviset na tom, zda je fronta `frBet` prázdná. Bude-li neprázdná, půjde `Hlidac` ihned do fronty, kde bude k disposici pro další obsluhu v betonárně. Bude-li fronta `frBet` prázdná, pak `Hlidac` projde nejprve pomocnou aktivitou `Pocitani`, kde se započítá jeho průchod a pak odejde do téže fronty, aby byl k disposici, jakmile se ve `frBet` objeví další auto. Model je znázorněn na obr. 7.5, str. 76.

```
entity  Auto;
entity  Hlidac;
```

```
queue   frBet    of Auto fifo;   fronta na betonárnu
queue   frCesta  of Auto fifo;   fronta na cestu s betonem
queue   frHlidac of Hlidac fifo;
queue   pom      of Hlidac fifo;  pomocná fronta pro počítání

activity  Betonarna  channels 1
  load 1 Auto   from frBet
       1 Hlidac from frHlidac
  after 5 ::  eject
    Auto to frCesta
    Hlidac to Vyh;
  end;

activity  Cesta  channels 0
  load 1 Auto from frCesta
  after dobaJizdy ::  eject
    Auto to frBet;
  end;

switch  Vyh  for  Hlidac
  case  contents(frBet) = 0  ::  Pom,
  otherwise                      frHlidac;

activity  Pocitani  channels 1
  load  1 Hlidac from Pom
  after 0.0 :: eject Hlidac to frHlidac;
  end;

declarations
  function  dobaJizdy : real;
    type  pole = array [1..7] of real;
    const  F : pole = (0.00, 0.06, 0.14, 0.52, 0.74, 0.91, 1.00);
           X : pole = (10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0);
    var  y : real;  i : integer;
  begin
    y:=random;
    i:=1;
    repeat  i:=i+1  until  F[i] >= y;
    dobaJizdy := x[i] - (F[i]-y) * (x[i]-x[i-1]) / (F[i]-F[i-1]);
  end;
::
end.
```

Fronty inicialisujeme příkazy `queue frBet 5` `queue frHlidac 1`. Simulací získáme tyto údaje:

```
---------- výpis statistik v čase   6000.000
pool  Auto            5=#i    0=#d   5=max   5.0000=μ# 6000.0000=μT
pool  Hlidac          1=#i    0=#d   1=max   1.0000=μ# 6000.0000=μT
que   frBet         916=#i  916=#d   5=max   0.3608=μ#    2.3633=μT
que   frCesta       915=#i  915=#d   1=max   0.0000=μ#    0.0000=μT
que   frHlidac      916=#i  916=#d   1=max   0.2372=μ#    1.5540=μT
que   pom           364=#i  364=#d   1=max   0.0000=μ#    0.0000=μT
act   Betonarna     916=#i  915=#d   1=max   0.7628=μ#    4.9962=μT
act   Cesta         915=#i  911=#d   5=max   3.8764=μ#   25.4193=μT
```

```
swtch Vyh           915=total#    364 --> pom
                                  551 --> frHlidac
act   Pocitani      364=#i  364=#d   1=max   0.0000=μ#    0.0000=μT
---------- konec výpisu statistik v čase   6000.000
```

Za dobu simulace 100 hodin betonárna čekala 364-krát na příjezd auta. Průměrně tedy čekala 3.64-krát za hodinu.

Nyní uvedeme druhý způsob získání týchž údajů. Deklarujeme globálně přístupnou proměnnou `pocetCekani` a využijeme toho, že entita `Auto` po opuštění aktivity `Betonarna` okamžitě vstupuje do aktivity `Cesta`, takže klausule `compute` se v aktivitě `Cesta` vykoná vždy ve stejném modelovém čase. Proto do klausule `compute` můžeme umístit test fronty `frBet` a zvyšování čítače `pocetCekani`. K inicialisaci čítače a k výpisu jeho hodnoty použijeme klausule `on start` a `on write`. Celý model pak je popsán takto:

```
entity  Auto;
queue   frBet    of Auto fifo;  { fronta na betonárnu }
queue   frCesta  of Auto fifo;  { fronta na cestu s betonem }

activity  Betonarna  channels 1
  load 1 Auto from frBet
  after 5 ::  eject Auto to frCesta;
  end;

activity  Cesta  channels 0
  load 1 Auto from frCesta
  compute
    if  contents (frBet) = 0  then begin
      pocetCekani:=pocetCekani+1;
      { writeln (Fou, 'Pocet cekani zvetsen na ', pocetCekani); }
      end;
    ::
  after dobaJizdy ::  eject Auto to frBet;
  end;

declarations
  var  pocetCekani : integer;

  function  dobaJizdy : real;
    type  pole = array [1..7] of real;
    const  F : pole = (0.00, 0.06, 0.14, 0.52, 0.74, 0.91, 1.00);
           X : pole = (10.0, 15.0, 20.0, 25.0, 30.0, 35.0, 40.0);
    var  y : real;  i : integer;
  begin
    y:=random;
    i:=1;
    repeat  i:=i+1  until  F[i] >= y;
    dobaJizdy := x[i] - (F[i]-y) * (x[i]-x[i-1]) / (F[i]-F[i-1]);
  end;
::

on start    pocetCekani := 0;  ::
on write    writeln (Fou, 'Počet čekání = ', pocetCekani);  ::

end.
```

Provedeme-li nyní stejnou simulaci jako výše, pak současně se statistickými údaji získáme v protokolu řádku

```
Počet čekání = 364
```

Poznamenejme, že při použití tohoto způsobu získávání informací o průběhu simulace je žádoucí věnovat zvýšenou pozornost ladění. Na místech, kde se informace sbírají, je dobré zapisovat do protokolu vhodné zprávy a tyto zprávy konfrontovat s ladicími informacemi systému DSI. V našem případě byl takto použit příkaz `writeln`, který je ponechán v klausuli `compute` aktivity `Cesta` jako komentář.

7.4.5 Cvičení. Pro betonárnu z předchozího příkladu určete minimální počet aut, při kterém betonárna vyrobí v průměru alespoň 10 dávek za hodinu.

7.4.6 Cvičení. Potřebujete postavit betonárnu, která by dodávala v průměru 10 dávek za hodinu. Vybíráte si ze dvou typů betonáren. Prvý stojí 6 milionů a vyrobí dávku za 3 minuty, druhý typ stojí 4 miliony a vyrobí dávku za 5 minut. Automobil pro přepravu betonu stojí 1 milion, jejich počet musíte zvolit podle typu betonárny tak, abyste dosáhli potřebného výkonu. Který typ betonárny bude výhodnější? Pro jednoduchost uvažujte pouze investiční náklady.

7.4.7 Cvičení. V obchodě obsluhují dva prodavači. Doba obsluhy je `2+negexp(4)` minut. Příchody zákazníků tvoří Poissonův proces s intensitou 15 zákazníků za hodinu. Je-li v obchodě fronta o více než 10 lidech, pak příchozí zákazník ihned odchází pryč. Průměrný zisk z jednoho zákazníka je 45 Kč. Majitel obchodu má možnost zrychlit obsluhu o 25 % zakoupením lepšího typu pokladny za 100 000 Kč. Je pravděpodobné, že se mu investice do pokladen vrátí dříve než po 1000 hodinách provozu?

7.4.8 Cvičení. Mějme jednokanálový systém hromadné obsluhy s Poissonovským vstupním procesem s intenzitou 8 zákazníků za hodinu. Doba obsluhy je 6 minut. Vlivem mimořádné události je nyní ve frontě 100 zákazníků. Zjistěte, za jak dlouho se systém vyprázdní.

Sestavit model takového systému je snadné. Trochu těžší je zjistit okamžik, kdy došlo k vyprázdnění. Navíc je žádoucí provést takových experimentů mnoho a vypočítat průměr. Jistě by bylo možné ručně měnit inicialisaci generátoru náhodných čísel a opakovat výpočty, ale bylo by to zbytečně pracné.

Navrhněte model, který bude fungovat takto: v okamžiku vyprázdnění systému se do fronty zařadí dalších 100 zákazníků a celý postup se bude opakovat. Podobně jako v příkladě 7.4.4, str. 74 můžete použijít entitu `Hlidac`, která se bude účastnit obsluhy spolu se zákazníky a která v okamžiku vyprázdnění systému způsobí vstup nové stovky zákazníků.

Hlídání, zda je systém prázdný, lze realisovat vyhýbkou podobně jako v příkladě 7.4.4, str. 74 a nebo pomocí klausule `signal to` v popisu fronty, kterou prochází hlídač. Proveďte obě varianty.

7.4.9 Cvičení. Navrhněte způsob, jak generovat skupiny entit o náhodné velikosti. Může jít například o situaci, kdy každých 30 minut přiveze autobus náhodný počet zákazníků.

7.4.10 Cvičení. Modelujte Q-systém řízení zásob. Předpokládejte, že skladujeme kusové zboží a každý zákazník požaduje jeden kus. Příchody zákazníků tvoří Poissonův proces s intensitou 6 zákazníků za hodinu. Poklesne-li zásoba pod signální úroveň, je objednáno další zboží.

Objednávka bude vyřízena po náhodné době 4–6 hodin s rovnoměrným rozložením. Zákazník, pro kterého nemáme zboží na skaldě, odchází ke konkurenci. Obsloužený zákazník představuje zisk 50 Kč. Skladovací náklady jsou 7 Kč za kus a hodinu. Každá objednávka představuje náklady 15 Kč. Najděte optimální velikost signální úrovně a optimální velikost objednávky.

7.4.11 Cvičení. Jedna ze starších versí systému DSI neměla funkci `contents` a klausuli `declarations`. Navrhněte způsob, jak v takto omezeném systému lze odeslat entitu do kratší ze dvou front.

Návod: použijte pomocnou entitu, která ponese informaci o obsazení fronty `Q` ve svém atributu. Tato entita se musí účastnit všech dějů, které ovlivňují počet entit ve frontě `Q`.

7.4.12 Cvičení. Ve starší versi systému DSI mohl popis aktivity obsahovat jen jednu klausuli `after`. Navrhněte způsob, jak za těchto podmínek modelovat aktivitu, která má klausule `after` dvě.

Nejprve zkuste řešit jednodušší případ, kdy jedna z klausulí určuje nulové zdržení, tj. má tvar `after 0.0 :: eject ...`

Promyslete obecný případ s náhodnými dobami, kdy není apriori jasné, která z obou dávek má být odeslána z aktivity dříve.

7.4.13 Cvičení. Představte si omezenou versi systému DSI, v níž není k disposici klausule `channels` a každá aktivita má neomezený počet kanálů. Navrhněte obecný způsob, jak za těchto podmínek modelovat omezený počet kanálů.

Návod: Použijte pomocnou entitu `volnyKanal`, která je potřebná ke spuštění aktivity a která aktivitu opustí až v poslední dávce. Všimněte si, že je-li dávek několik, pak je třeba předem vědět, která dávka bude odesílána jako poslední. Počet kanálů se určí při inicialisaci fronty volných kanálů.

Výsledky

V–svsec

Entita typu `Opravar` bude vstupovat do aktivity `Oprava` ze své fronty `qOpravar` a po okončení opravy se do této fronty vrátí. Počet opravářů v systému bude tedy konstantní a bude dán počátečním obsazením fronty `qOpravar`. Počet kanálů aktivity `Oprava` přitom může být neomezený, neboť počet současně probíhajících oprav je omezen počtem opravářů. Ve všech výpočtech tedy lze použít stejný model a stejný simulační program. Jednotlivé výpočty se liší pouze v inicialisaci fronty `qOpravar`.

V–svsec

Základem řešení je vyhodnocení jednotlivých variant. Pro vyhodnocení varianty můžete aktivitu `Oprava` nahradit několika (až čtyřmi) aktivitami s dobani oprav podle tabulky. Jednotlivé varianty se budou lišit v počtech kanálů. Případné priority pro obsazování jednotlivých opravářů lze určit v popisu fronty klausulí `signal to`.

Druhou možností je použít entitu typu `Opravar` jako ve cvičení 7.4.2, str. 73. Tato entita bude mít atribut (nebo dva atributy), podle kterých se určí (tj. náhodně vygeneruje) doba opravy. Pro výpočet jednotlivé varianty pak je nutno inicialisovat frontu `qOpravar` včetně hodnot atributů jednotlivých opravářů. Priority pro obsazování opravářů lze určit pomocí režimu fronty `qOpravar`.

V–svsec

Zjistěte simulací průměrný počet obsloužených zákazníků za hodinu pro obě varianty pokladen. Odtud lze zjistit průměrný zisk za hodinu. Pokud s novými pokladnami bude zisk vyšší alespoň o 100 Kč za hodinu, pak se investice vrátí dříve než za 1000 hodin provozu.

Literatura

[1] Klvaňa, J.: *Operační výzkum I*, skriptum ČVUT, Fakulta stavební, 1990

[2] Klvaňa, J.: *Operační výzkum II*, skriptum ČVUT, Fakulta stavební, 1993

[3] Dudorkin, J.: *Operační výzkum*, skriptum ČVUT, Fakulta elektrotechnická, 1990

[4] Demel, J.: *Grafy*, SNTL Praha, 1988

[5] Kučera, L.: *Kombinatorické algoritmy*, SNTL Praha, 1983

[6] Morávek, J.: *O dynamickém programování*, Mladá fronta, 1973

[7] Zítek, F.: *Ztracený čas, Elementy teorie hromadné obsluhy*, Academia Praha, 1969

[8] Shanon, R. E.: *Systems Simulation, the Art and Science*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1975

Obsah