Trasování ve struktuře

možnosti:

- přímo ve struktuře s využitím metod pro konektivitu portů:

  příklad: (rekapitulzce):

  ```
  cGate* pout0 =  gate ( “out” , 0);
  Gate*    p_pripojeny_port        =  pout0 -> toGate();
  cModule* p_pripojeny_module =
                                  p_pripojeny_port ->ownerModule();
  ```

- v pomocném grafu, který lze použít jako podporu směrování, protože také umožňuje výpočet vzdáleností mezi zvolenými uzly;

  princip podpory směrování:

  - objekt třídy cTopology představuje grafovou reprezentaci struktury sítě z vybraných modulů,
  - vytvoření grafu: výběrem zvolených modulů simulované sítě,
  - uzly (tj.objekty podtřídy cTopology :: Node) reprezentují moduly (jednoduché i složité),
  - orientované hrany (tj.objekty podtřídy cTopology :: Link) reprezentují komunikační linky (neohodnocené),
  - Dijkstrův algoritmu nad vytvořeným grafem slouží pro výpočet nejkratších cest do určitého uzlu,
  - na základě korespondencí mezi moduly a uzly grafu lze výsledek Dijkstrova algoritmu (uložený v objektu třídy cTopology) aplikovat na vzdálenosti mezi moduly .

Metody pro podporu směrování

Metody třídy cTopology:

a) metody pro vytvoření reprezentujícího grafu:

- void cTopology :: extractByModuleType (const char * type,….)
  - vytvoří neohodnocený orientovaný graf reprezentující vazby mezi moduly vybranými pomocí seznamu typů:

    příklad:

    cTopology graf; // grafová reprezentace sítě

    graf . extractByModuleType (“ Router1“, “ Router2“ , NULL);

    poznámky:
  - seznam typů musí končit hodnotou NULL,
  - jako aktuální parametr lze použít ukazatel na pole pointrů

- void cTopology :: extractByParametr

  (const char * parname, cPar * value)
  - generuje graf z modulů vlastnících daný parametr (a s případnou hodnotou),

    příklad:

    cTopology graf;

    graf . extractByModuleType (“ IPadresa“);

- int cTopology :: nodes ( ) ….vrací celkový počet uzlů grafu,
- cTopology :: Node* :: node (int i) ..vrací pointr na i-tý uzel grafu

Metody pro podporu směrování

b) metody pro zjištění nejkratší cesty:

- void cTopology :: unweightedSingleShortestParthsTo
  (cTopology :: Node * p)

  pro každý uzel grafu provede výpočet nejkratší cesty z tohoto uzlu do cílového uzlu (odkazovaného aktuálním parametrem),

- poznámka: připravují se další funkce (např. pro nalezení cest v grafu s ohodnocenými hranami) ,

- cTopology :: Node *cTopology :: targetNode ( )

  vrací ukazatel na cílový uzel posledního hledání nejkraší cesty

c) metody pro mapování modul → uzel:

- cTopology :: Node *cTopology :: nodeFor (cModule *mod)

  vrací ukazatel na uzel reprezentující v grafu modul odkazovaný parametrem mod; není-li daný modul reprezentován žádným uzlem grafu pak vrací hodnotu NULL

Metody pro podporu směrování

Metody třídy cTopology :: Node:

a) metoda pro mapování uzel → modul:

- cModule * cTopology :: Node :: module ( )

  vrací ukazatel na modul reprezentovaný daným uzlem grafu,

b) metody pro zjišťování vzdálenosti:

- int cTopology :: Node :: distanceToTargit ()

  vrací vzdálenost vypočtenou při předchozím volání funkce unweightedSingleShortestParthsTo( ) ; tato vzdálenost je vyjádřená počtem hran z daného uzlu do cílového uzlu;

- int cTopology :: Node :: paths ( )

  při volání funkce unweightedSingleShortestParthsTo (..) vrací pouze hodnotu 1 (v případě nalezení cesty) nebo 0 (přinenalezení cesty); jinak vrací počet cest z daného uzlu do cílového uzlu,

- cTopology :: Link* :: cTopology :: Node :: path (int i)

  při volání funkce unweightedSingleShortestParthsTo (..) jde o jedinou cestu a odkaz na příslušnou hranu vrací path(0); jinak vrací odkaz na hranu daného uzlu, která reprezentuje i-tou nejkratší cestu do cílového uzlu;

Metody pro podporu směrování

c) metody pro trasování v grafu:

- int cTopology :: Node :: inLinks ()

  vrací počet vstupních hran daného uzlu,

- int cTopology :: Node :: outLinks ()

  vrací počet výstupních hran daného uzlu,

- cTopology :: Link *cTopology :: Node :: in (int i)

  vrací ukazatel na i-tou vstupní hranu daného uzlu (objekt třídy cTopology :: Link),

- cTopology :: Link *cTopology :: Node :: out (int i)

  vrací ukazatel na i-tou výstupní hranu daného uzlu (objekt třídy cTopology :: Link),

d) metody pro změnu stavu uzlu:

- void cTopology :: Node :: disable ( )

  uvede daný uzel do stavu ve kterém je ignorován při výpočtu nejkratší cesty,

- void cTopology :: Node :: enable ( )

  uvede daný uzel do stavu ve kterém je při výpočtu nejkratší cesty uvažován,

Metody pro podporu směrování

Metody třídy cTopology :: Link:

- cTopology :: Node *cTopology :: Link :: remoteNode ( )
  vrací ukazatel na uzel připojený na vzdálený konec dané hrany,

- mapování hrana → port:

- cGate *cTopology :: Link :: localGate ( )
  vrací ukazatel na lokální port reprezentovaný danou hranou,
- cGate *cTopology :: Link :: remoteGate ( )
  vrací ukazatel na vzdálený port reprezentovaný danou hranou,
- int *cTopology :: Link :: localGateId ( )
  vrací identifikaci lokálního portu reprezentovaného danou hranou,
- int *cTopology :: Link :: remoteGateId ( )
  vrací identifikaci vzdáleného portu reprezentovaného danou hranou,

Příklad trasování

Vyhledání všech následovníků modulu určeného pointrem ptr_m pomocí grafu

```
cTopology graf ;              // deklarace objektu pro vytvoření grafu
cModule*  ptr_m =  this;   // pointer na tento module
// následuje vytvoření grafové reprezentace vybraných modulů
graf . extractByModuleType ( " Tmodule1" , NULL );
 cTopology :: Node * pu1 = graf . nodeFor ( ptr_m ) ;
 for ( int j = 0 ; j < pu1 . outLinks( ); j++ )
     {
       // cyklus přes  všechny výstupní  hrany uzlu pu1
       cTopology :: Link * ph = pu1 -> out ( j );
       // ph = pointr na j-tou hranu uzlu pu
       cTopology :: Node * pu2 = ph -> remoteNode ( );
       // pu2 = pointr na sousední uzel připojený na j - tou
       // hranu daného  uzlu  pu1
       cModule * pm_nasl20 = pu2 -> module ( );
       // pm_nasl20 = odkaz na modul reprezentovaný  uzlem pu2

       // následuje jiný způsob nalezení následníků modulu  ptr_m
       cGate * pport2 = pu1  -> out ( j ) -> localGate (  );
       // pport2 ..odkaz na port spojující modul ptr_m s modulem
       // pm_nasl2
       cModule* pm_nasl21= pport2 -> toGate() ->ownerModule();
     }
```

Příklad: nejmenší vzdálenost mezi moduly

Určení výstupního portu vport modulu pzm, který leží na nejkratší cestě vedoucí od modulu pzm k modulu pcm

```
cTopology  grf ;
gfr . extractByModuleType(……., NULL);

// určení pointru pzu na uzel reprezentující  zdrojový modul pzm:
cTopology :: cNode *  pzu = grf ->  nodeFor ( pzm):

// určení pointru pcu na uzel reprezentující cílový modul pcm:
cTopology :: cNode *  pcu = grf ->  nodeFor ( pcm):

// výpočet nejkratších cest ze všech uzlů grafu grf do uzlu pcu:
grf . unweightedSingleShortestPathsTo ( pcu );
ev << "delka nejkratsi cesty do ciloveho uzlu:"
     << pzu -> distanceToTarget () << endl;

cGate* vport = pzu -> path(0) ->localGate();    // hledany port
ev << „ nejkratší cesta od  pzm k pcm vede  přes port:   "
    << vport -> fullPath() << endl;
```

Příklad: nejmenší vzdálenost mezi moduly

Jiné řešení minimální cesty : bez použití metody path(0)

```
 cTopology  grf ;   gfr . extractByModuleType(……., NULL);
 // určení pointru pzu na uzel reprezentující  zdrojový modul pzm:
 cTopology :: cNode *  pzu = grf ->  nodeFor ( pzm):
// určení pointru pcu na uzel reprezentující cílový modul pcm:
 cTopology :: cNode *  pcu = grf ->  nodeFor ( pcm):
pzu -> disable ( );    // vyjmutí uzlu pzu z grafu grf:
 // výpočet nejkratších cest ze všech uzlů grafu grf do uzlu pcu:
 grf . unweightedSingleShortestPathsTo ( pcu );
 pzu -> enable ( );  // znovuzačlenění uzlu  pzu do grafu grf:
// nalezení  minimální vzdáleností mezi všemi následníky uzlu pzu
    // a cílovým uzlem pcu:
 int  min = MAX;      // počáteční nastavení velké hodnoty
cGate* vport;             // ptr na hledaný port
 for ( int i = 0; i < pzu . outLinks(); i++ );
     { cTopology :: Link * phr = pzu->out ( i ); // ptr na i-tou hranu
       cTopology :: Node * nasl = phr -> remoteNode ( );
       if ( nasl -> paths (  ) = 0 )   continue;    // tudy cesta nevede,
       if ( nasl -> distanceToTargit ( ) <  min)
                                {  min = nasl -> distanceToTargit ( ) ;
                                   vport = phr-> localGate();     }
     }
ev << „ nejkratší cesta od  pzm k pcm vede  přes port:   "
    << vport -> fullPath() << endl;
```

Příklad: analýza propojení

Uzel typu Router analyzuje propojení sítě , zjišťuje následující:

- korespondenci mezi moduly sítě a uzly grafu,
- počty vstupních i výstupních hran jednotlivých uzlů,
- porty, přes které jsou jednotlivé moduly propojeny.

```
simple Router gates:  in: in[ ];    out: out[ ];    endsimple
simple Stanice  gates:  in: in[];   out: out[];  endsimple
module SIT
    submodules:
        r:  Router ;        gatesizes:   in [3], out [3];
        s: Stanice [8];   gatesizes:  in [3], out [3];
    connections nocheck:    // neúplně propojená síť
        r.out[0]     --> s[1].in[0];
        r.out[1]     --> s[2].in[0];
        s[1].out[0] --> s[4].in[0];
        s[2].out[0] --> s[3].in[0];
        s[3].out[0] --> s[4].in[1];
        r.out[2]     --> s[5].in[0];
        s[5].out[0] --> s[6].in[0];
        s[6].out[0] --> s[7].in[0];
        s[7].out[0] --> s[4].in[2];
 endmodule
network sit: SIT
endnetwork
```

Příklad: analýza propojení

```
void Router::initialize()
{
  cTopology  graf;
  // následuje  vytvoreni grafu
  graf . extractByModuleType ( "Router", “Stanice“,  NULL );
  for ( int i = 0; i < graf . nodes() ; i++ )  // cyklus pres vsechny uzly
     { cTopology :: Node* uzel = graf . node (i); // ptr na i-ty uzel
       ev << "uzel=  " << i << " reprezentuje modul "
          << uzel -> module() -> fullPath() << endl;
       ev << "pocet vstupnich hran = "  << uzel -> inLinks()
          <<  " pocet vystupnich hran = " << uzel -> outLinks()
          << endl;
       ev << " instance pripojene na vystup uzlu:  \n";
       for ( int j = 0; j < uzel-> outLinks(); j++)
            {
              cTopology :: Node* sousedni_uzel =
                    uzel -> out (j)  -> remoteNode();
             ev <<  "pres port:   "
                <<   uzel -> out (j) ->localGate()  ->  fullPath()
                <<  "  pripojen uzel    "
                <<  sousedni_uzel   -> module() -> fullPath()
                <<   "  portem  "  << uzel -> out (j) ->remoteGate()
                <<   endl;
            }
}
```

Příklad: analýza propojení – výstupy

uzel = 0 reprezentuje modul sit.r
pocet vstupnich hran = 0 pocet vystupnich hran = 3
instance pripojene na vystup uzlu:
přes port: sit.r.out[0] pripojen uzel sit.s[1] portem (cGate)in[0]
pres port: sit.r.out[1] pripojen uzel sit.s[2] portem (cGate)in[0]
pres port: sit.r.out[2] pripojen uzel sit.s[5] portem (cGate)in[0]
……

uzel= 5 reprezentuje modul sit.s[4]
pocet vstupnich hran = 3 pocet vystupnich hran = 0
instance pripojene na vystup uzlu:

……..

uzel= 7 reprezentuje modul sit.s[6]
pocet vstupnich hran = 1 pocet vystupnich hran = 1
instance pripojene na vystup uzlu:
pres port: sit.s[6].out[0] pripojen uzel sit.s[7] portem (cGate)in[0]

uzel= 8 reprezentuje modul sit.s[7]
pocet vstupnich hran = 1 pocet vystupnich hran = 1
instance pripojene na vystup uzlu:
pres port: sit.s[7].out[0] pripojen uzel sit.s[4] portem (cGate)in[2]

Příklad: přepínač se směrovacími tabulkami

```
// funkční moduly
simple Generator  gates:  out: out;        endsimple
simple Likvidator   gates:  in:   in;          endsimple
simple Prepinac    gates:  in: ing, inp; out: outl, outp; endsimple

simple SIT_vrstva
      parameters:   pocet: numeric;
      gates:  in: in [ ];  inh[ ];   out: out[ ], outh[ ];   endsimple

simple Server        gates:    in:  in;   out:  out;      endsimple
// vnitřní struktura počítače
module Pocitac
   parameters:    adresa, celk_pocet: numeric;
   gates:              in:  inp;      out: outp;
   submodules:
        gen: Generator;
        lik:    Likvidator;
        pr:    Prepinac;
   connections:
        gen.out  --> pr.ing;
        pr.outl    --> lik.in;
        inp          --> pr.inp;
        pr.outp    --> outp;
endmodule
```

Příklad: přepínač se směrovacími tabulkami

```
// vnitřní struktura směrovače:
module Smerovac
    parameters: adresa, pocet_h : numeric;
    gates: in: vst [ ], vstp [ ];  out: vyst[ ], vystp[ ];
    submodules:
      sit:     SIT_vrstva ;
               parameters: pocet = pocet_h;
               gatesizes: in [2], inh[pocet_h],  out[2], outh[pocet_h];
      server: Server [ pocet_h + 2 ];
    connections:
    // propojeni  instance sit a instancí server s porty k počítačům
      for i = 0 .. pocet_h - 1 do
        vstp[i]          -->  sit.inh [i];        // linka od počítače
        sit.outh[i]      -->  server[i].in;     // linka od síť vrstvy k serveru
        server [i].out --> datarate 100 --> vystp [i];  // linka k počítači
      endfor
      // propojeni  instance sit a instancí server s porty dvou
      // sousedních  směrovačů:
      for i = 0 .. 1 do
        vst [i]                  -->  sit.in [i];
        sit.out [i]              -->  server [pocet_h + i].in;
        server [pocet_h + i].out --> datarate 100 --> vyst [i];
      endfor;
endmodule
```

Příklad: přepínač se směrovacími tabulkami

```
// celková struktura sítě
module SIT
   parameters:
                    pocet_poc, pocet_smer:   numeric;
   submodules:
    poc:   Pocitac [pocet_smer * pocet_poc];
           parameters: adresa = index,  // adresa  instance poc
                       celk_pocet = pocet_poc;  // pro adresování

    smer:  Smerovac [pocet_smer];
           parameters: adresa = index + pocet_smer * pocet_poc,
                       pocet_h = pocet_poc;
           gatesizes: vst [2], vyst [2], vstp [pocet_poc],
                      vystp [pocet_poc];

connections:
 // následuje vzájemné propojeni směrovačů
for i = 0 .. pocet_smer - 2 do
      smer[i].vyst[1]      --> datarate 100  -->  smer[i+1].vst[0];
      smer[i+1].vyst[0]   --> datarate 100  --> smer[i].vst[1];
endfor;
smer[pocet_smer - 1].vyst[1]  --> datarate 100 --> smer[0].vst[0];
smer[0].vyst[0]  --> datarate 100 --> smer[pocet_smer - 1].vst[1];
```

Příklad: přepínač se směrovacími tabulkami

```
//  následuje propojení směrovačů a  přidružených počítačů
     smer[0].vystp[0]  -->  poc[0].inp;
    poc[0].outp       -->  smer[0].vstp[0];
    smer[0].vystp[1]  -->  poc[1].inp;
    poc[1].outp       -->  smer[0].vstp[1];

    smer[1].vystp[0]  -->  poc[2].inp;
    poc[2].outp       -->  smer[1].vstp[0];
    smer[1].vystp[1]  -->  poc[3].inp;
    poc[3].outp       -->  smer[1].vstp[1];

    smer[2].vystp[0]  -->  poc[4].inp;
    poc[4].outp       -->  smer[2].vstp[0];
    smer[2].vystp[1]  -->  poc[5].inp;
    poc[5].outp       -->  smer[2].vstp[1];
endmodule                 // konec struktury sítě
 // instance sítě:
network sit :  SIT       // výsledná síť
endnetwork

// vnitřní struktura paketů:
message PAKET
{   fields:   int   Zdr_adr;
              int  Cil_adr;       } ;
```

Příklad: přepínač se směrovacími tabulkami

```
#include <omnetpp.h>
#include <map>
#define STACKSIZE 16384
class Generator : public cSimpleModule
{ public:  virtual void activity();  Module_Class_Members ( …)  } ;

class Likvidator : public cSimpleModule
{  public:   virtual void handleMessage(cMessage *msg);  ….  };

class SIT_vrstva : public cSimpleModule
{ private:  typedef std:: map < int, int> Smerovaci_tabulka;
            Smerovaci_tabulka  smer_tab;
  public:  virtual void initialize();
  virtual void handleMessage(cMessage *msg);
  Module_Class_Members (…..)  };

class Server : public cSimpleModule
{  public:   cQueue fr;    cMessage endSend;
    virtual void handleMessage(cMessage* msg);……. };

class Prepinac : public cSimpleModule
{  public:  virtual void handleMessage(cMessage *msg);….. };
```

Příklad: přepínač se směrovacími tabulkami

Funkční popisy vybraných modulů:

```
void Generator::activity ()
{ cModule* poc = parentModule();  // ptr na počítač
  int pocetp  = poc ->par("celk_pocet"); // pocet pocitacu na 1 sm
  int pocetsm  = poc -> parentModule()-> par("pocet_smer");
  int pocetc     = pocetp * pocetsm;       // celk. pocet pocitacu v siti
  int adr   = poc -> par ("adresa");       // adresa daneho pocitace
  ev << "pocitac s adresou " << adr <<
          " vysila pakety vsem pocitacum v siti" << endl;
  for ( int i = 0; i < pocetc; i++ )
          { PAKET *p = new PAKET( "paket" );
             p -> setZdr_adr (adr);
             p -> setCil_adr (i);
             p -> setLength(…..);       // nastavení délky paketu
             ev << "paket ze zdroje s adresou  " << adr
                   << " je vyslan na adresu " << i << endl;
             send( p, "out" );
          }
}
void Likvidator::handleMessage (cMessage *msg)
{  PAKET * p = check_and_cast <PAKET*> (msg);
    ev << “paket od zdroje s adr.  " << p -> getZdr_adr()
       <<  “v cili s adr “ << parentModule() -> index()  <<  endl;
   delete msg;
}
```

Příklad: přepínač se směrovacími tabulkami

```
void Prepinac::handleMessage (cMessage *msg)
{
  PAKET * p = check_and_cast <PAKET*> (msg);
  if ( p-> arrivedOn  ("ing")) send (p,"outp");  // od generátoru k sm.
                    else  send (p,"outl");              // od sm k likvidátoru
}



void Server::activity()
{
      cMessage* msg;
 W: msg = receive();              // cekani na ramec
 S: send (msg, "out");
    cBasicChannel * k = check_and_cast<cBasicChannel*>
                              (gate("out") ->channel());  //pointer na kanal
    double del = msg->length()/k -> datarate(); //vypocet vys. doby
    waitAndEnqueue ( del, &fr);    // pozastaveno behem vysilani
    if  (! fr.empty())  {
                             msg = (cMessage*) fr. pop();
                             goto S;  //  pakety cekaji ve fronte
                            }
                  else goto W;    // není co vysilat
}
```

Příklad: přepínač se směrovacími tabulkami

```
void SIT_vrstva::initialize()     // vytvoreni smerovaci tabulky
{ cTopology* sit = new cTopology ("sit");
  sit -> extractByModuleType ("Pocitac", "Smerovac", NULL);
  // nasleduje odkaz na uzel reprezentujici tento Smerovac:
  cTopology ::Node *tento_uzel = sit->nodeFor ( parentModule() );
 // cyklus pro vyhledání nejkratších cest z daného uzlu  do
 // vsech zbývajících  uzlů site :
 for ( int i = 0; i < sit -> nodes(); i++)
    {  if ( sit -> node (i) == tento_uzel ) continue;  // i = cilovy uzel
       sit -> unweightedSingleShortestPathsTo (sit -> node(i));
       if ( tento_uzel -> paths () == 0) continue; // cesta není
       cGate * vystup_port_smerovace = tento_uzel -> path(0)
                                                  ->localGate();
     // nasleduje hledani vystupniho portu sitove vrstvy:
     // nejprve vypocet pointru na server
     cModule * ptr_na_server = vystup_port_smerovace ->
                                       fromGate() -> ownerModule();
     // nasleduje urceni identifikace   vyst. portu sit vrstvy
     int ident_portu_sit_vrstvy = ptr_na_server -> gate ("in") ->
                                        fromGate () -> id();
     // nasleduje  zjištění adresy cílového modulu a vyplnění jedné
              // položky směrovací tabulky
     int adresa_cile = sit -> node(i) -> module() -> par ("adresa");
     smer_tab [adresa_cile]  =  ident_portu_sit_vrstvy;
   }
```

Příklad: přepínač se směrovacími tabulkami

```
 delete sit;        // uloha pomocneho grafu splnena
// následuje kontrolní tisk směrovací tabulky  daného směrovače
 int adr = parentModule() -> par ("adresa");   // adresa  smerovace
 ev  << "smerovaci tabulka smerovace s adresou " <<  adr <<
            “ je nasledujici: ”  << endl;
Smerovaci_tabulka :: iterator  iter;
 for ( iter = smer_tab.begin();  iter != smer_tab.end();  iter++ )
       { ev << " cilovemu pocitaci s adresou“   << (*iter) . first  <<
                  " je treba posilat zpravy pres port s identifikaci “
.                   << (*iter) . second <<  endl;
       }
}    end of initialize ()

void SIT_vrstva::handleMessage(cMessage *msg
{   PAKET * p = check_and_cast <PAKET*> (msg);
    Smerovaci_tabulka :: iterator iter = smer_tab.find ( p ->
                                                    getCil_adr() ) ;
    if ( iter == smer_tab.end())
                        { ev << "nedorucitelny paket" << endl;
                          delete p;  return;   }
    // nasleduje vyber spravneho portu ze smerovaci tabulky a
   // odeslani prijateho paketu
     send (p, (*iter).second );
}   end of handleMessage()
```

Dynamické vytváření struktur

OMNET++ umožňuje:

- generovat instance modulů jednotlivých typů při výpočtu,
- propojovat, případně odpojovat instance modulů při výpočtu

Funkce pro generování složených i jednoduchých modulů:
pro každý typ modulu existuje objekt - generátor (třídy cModuleType), který může dynamicky generovat instance daného typu.

cModuleType* cModule :: findModuleType (const char * *M*)
…..vrátí pointr na generátor instancí typu M,

virtual cModule* cModuleType :: cCreateScheduleInit
(const char "jm", cModule * rod);
….vytvoří instanci konkrétního typu (zde M) včetně parametrů a portů se jménem jm a rodičem rod, zajistí provedení funkce Initialize() této instance a vrátí pointer na tuto instanci. Pokud se vytváří složený modul, jehož struktura závisí na dosud neurčených parametrech a dimenzích portů, pak výše zmíněnou funkci je třeba rozložit do více kroků, které lze zajistit pomocí dalších funkcí (podrobnosti viz manuál):

virtual cModule* cModuleType ::create (const char "jm",
cModule * rod);

int cModule:: setGateSize (const char *s, int size)

virtual void cModule:: buildInside( )

virtual void cModule:: schedule Start (simtime_t t=0)

virtual void cModule:: callInitialize ()

Dynamické vytváření struktur

Funkce pro vytváření spojení mezi dynamicky generovanými moduly:

void cGate :: connectTo (cGate* ptrg, cChannel* ptrch = NULL)

……vytvoří spojení od výstupního portu ke vstupnímu portu (tj.portu specifikovanému parametrem ptrg a přiřadí takto vytvořenému spojení parametry kanálu ptrch; v případě dříve vytvořeného spojení hlásí chybu,

void cGate :: disconnect ()

……zruší spojení od instance výstupního portu (jehož metoda je volána) k vstupnímu portu připojeného modulu,

void cGate :: setChannel (cChannel* ptrch)

……přiřadí specifikovaný kanál k danému portu

Funkce pro rušení modulů:

void cModule :: deleteModule ( );

- v případě složeného modulu se zruší všechny dílčí moduly,
- tato metoda nevolá funkci finish ( ) zrušeného modulu,
- funkci finish ( ) rušeného modulu lze volat funkcí void cModule :: callFinish ( ) (včetně funkcí finish ( )) všech dílčích modulů,

Příklad: dynamicky vytvořená sběrnicová síť

- instance gen modulu Dynasit vygeneruje jednu instanci bus modulu Bus a posléze dvě instance (st0, st1) modulu Stanice, které připojí ke sběrnici,
- instance st0 i st1 po své inicializaci pošlou zprávu na sběrnici,
- sběrnice (zde nejjednodušší možná a umožňující připojení pouze dvou stanic) po přijetí zprávy přepošle přijatou zprávu další stanici pokud tato je připojena; jinak zprávu zahodí.

Soubor celek.ned **:**

simple Generator // generator stanic
endsimple

simple Stanice
gates: **in**: inp; **out**: outp; **endsimple**

simple Bus
gates: **in**: in0, in1; **out**: out0, out1; **endsimple**

module Dynasit
submodules: gen: Generator; // staticky deklarovaná instance
connections nocheck: // potlačí kontrolu zapojení
endmodule

network dynasit: Dynasit
endnetwork

Příklad: dynamicky vytvořená sběrnicová síť

Soubor celek.cpp:

```
#include <omnetpp.h>

class Generator : public cSimpleModule
 {  public:
    virtual void activity();
    Module_Class_Members(Generator,cSimpleModule,32768) };

class Stanice : public cSimpleModule
 {  public:
    cMessage* start;  // ptr na vlastní zprávu
    virtual void initialize();
    virtual void handleMessage(cMessage * msg);
    virtual void finish();
    Module_Class_Members( Stanice,cSimpleModule,0)
 };

class Bus : public cSimpleModule
 {  public:
    virtual void initialize();
    virtual void handleMessage(cMessage * msg);
    Module_Class_Members(Bus,cSimpleModule,0)
 };
```

Příklad: dynamicky vytvořená sběrnicová síť

```
void Generator::activity()
{ // následuje vytvoření modulu bus
  cModuleType* gen_bus = findModuleType ("Bus");
  cModule* pbus = gen_bus -> createScheduleInit ( "bus", this );
  ev << "sbernice vytvorena" << endl;
  // následuje vytvoření stanice 0
  cModuleType* gen_st = findModuleType("Stanice");
  cModule* pst0 = gen_st -> create ( "st0", this );// ptr na stanici 0
  // následuje vytvoření spojení od stanice  ke sběrnici
  pst0 ->gate("outp") -> connectTo ( pbus -> gate ("in0"));
  // následuje vytvoření spojení od sběrnice ke stanici
  pbus -> gate("out0") -> connectTo ( pst0 ->gate("inp" ));
  ev << "stanice" << pst0->id()
     << " je vytvorena a pripojena v case:  " << simTime() << endl;
  pst0->callInitialize();  // inicializace stanice
  wait ( 10 );
  // následuje vytvoření modulu stanice 1
  cModule* pst1 = gen_st -> create ( "st1", this );// ptr na stanici 1
  pst1->gate("outp") -> connectTo ( pbus -> gate ("in1"));
  pbus -> gate("out1") -> connectTo ( pst1->gate("inp" ));
  ev << "stanice" << pst1-> id()
     << " je vytvorena a pripojena v case:  " << simTime() << endl;
  pst1->callInitialize();
  wait (20);
```

Příklad: dynamicky vytvořená sběrnicová síť

```
// následuje odpojení obou stanic
 pst0 ->gate("outp")->disconnect();
 pst1 ->gate("outp")->disconnect();
 pbus ->gate("out0")->disconnect();
 pbus ->gate("out1")->disconnect();

// následuje test odpojení
 if(! pst0->gate("outp") ->isConnected())
                    ev << "vystup outp nepripojen" << endl;
 if(! pst0->gate("inp")  ->isConnected())
                    ev << "vstup inp nepripojen" << endl;
 ……..;
 if(! pbus->gate("out0") ->isConnected())
                    ev << "vystup out0 nepripojen" << endl;
 if(! pbus->gate("out1") ->isConnected())
                     ev << "vystup out1 nepripojen" << endl;

// následuje provedení příslušných funkcí finish()
 pst0 -> callFinish();      // volá funkci finish() všech submodulů
 delete pst0;
 pst1 -> callFinish();
 delete pst1;

}            // konec procesu activity()
```

Příklad: dynamicky vytvořená sběrnicová síť

```
void Stanice::initialize()
{  ev << "stanice" << id() <<" se inicializuje" << endl;
   start = new cMessage( "start" );
   // následuje test připojení a zahájení vlastní komunikace
   if (gate("outp") -> isConnected() )
                                   scheduleAt( simTime()+ 1, start );     }

void Stanice::handleMessage(cMessage *msg)
{  if ( msg -> isSelfMessage () )
       { msg = new cMessage ( "z1"); msg -> setKind ( id() );
         send ( msg, "outp" );
         ev << "stanice  "  <<  id() <<  "odeslala zpravu pres port: "
              << gate ( "outp“ ) -> fullPath() << " v case:  "
              << simTime() << endl;
      } else ev << "stanice" << id ()
                   <<" obdrzela zpravu od stanice:  "
                << msg -> kind() << " v case:  " << simTime() << endl; }

void Stanice :: finish()
{       ev << "stanice " << id() << " zrusena v case:"
             << simTime() << endl;          }

void Bus::initialize()
{       ev << "bus se inicializuje" << endl;        }
```

Příklad: dynamicky vytvořená sběrnicová síť

```
void Bus::handleMessage(cMessage *msg)
{   if ( msg -> arrivedOn ("in0"))  // test příchozího portu
    // následuje jiná možná verse testu
    // if (gate ("in0")-> isConnected())
    // if ( gate ("in0")->fromGate()->ownerModule() ==
                                    simulation . module ( msg -> kind()))
     {  ev << "sbernice prijala zpravu od stanice: " <<  msg ->kind()
           << " a posle ji pripojene stanici"<< endl;
        if ( gate("out1") -> isConnected())
                              {  send ( msg, "out1");
                                 ev << "odeslano dalsi stanici" << endl;
                              }  else {  delete msg;
                                         ev << "zprava zahozena" << endl; }
     }
    if ( msg -> arrivedOn ("in1"))
     {  ev << "sbernice prijala zpravu od stanice: " << msg -> kind()
           << " a posle ji pripojene stanici"<< endl;
        if ( gate("out0") ->isConnected())
                              {  send ( msg, "out0");
                                 ev << "odeslano dalsi stanici" << endl;
                              }  else { delete msg;
                                         ev << "zprava zahozena" << endl; }
     }
}       // konec handleMessage
```

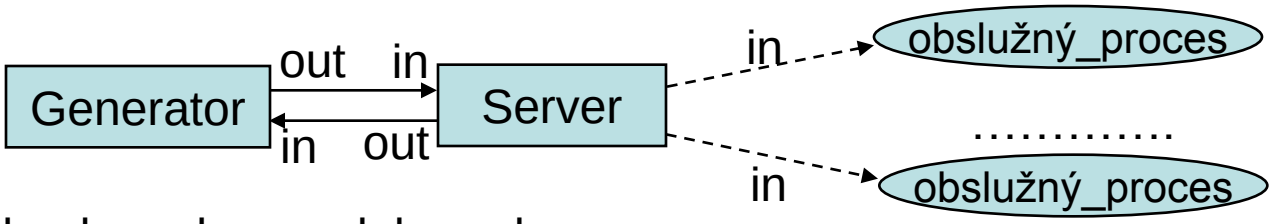
Příklad: dynamicky vytvořená sběrnicová síť

Výsledná hlášení jednotlivých modulů:

```
** Event #0.  T=0.0000000  ( 0.00s).  Module #2 `dynasit.gen'
sbernice vytvorena
bus se inicializuje
stanice4 je vytvorena a pripojena v case:  0
stanice4 se inicializuje
** Event #1.  T=       1  ( 1.00s).  Module #4 `dynasit.gen.st0'
stanice  4odeslala zpravu pres port: dynasit.gen.st0.outp v case:  1
** Event #2.  T=       1  ( 1.00s).  Module #3 `dynasit.gen.pbus'
sbernice prijala zpravu od stanice: 4 a posle ji pripojene stanici
zprava zahozena sbernici
** Event #3.  T=      10  (10.00s).  Module #2 `dynasit.gen'
stanice5 je vytvorena a pripojena v case:  10
stanice5 se inicializuje
** Event #4.  T=      11  (11.00s).  Module #5 `dynasit.gen.st1'
stanice  5odeslala zpravu pres port: dynasit.gen.st1.outp v case:  11
** Event #5.  T=      11  (11.00s).  Module #3 `dynasit.gen.pbus'
sbernice prijala zpravu od stanice: 5 a posle ji pripojene stanici
odeslano dalsi stanici
** Event #6.  T=      11  (11.00s).  Module #4 `dynasit.gen.pst0'
stanice4 obdrzela zpravu od stanice:  5 v case:  11
** Event #7.  T=      30  (30.00s).  Module #2 `dynasit.gen'
vystup outp nepripojen
vstup inp nepripojen
.....
vystup out1 nepripojen
stanice 4 zrusena v case:30
stanice 5 zrusena v case:30
```

Příklad: generování obslužných procesů

- server pro každou novou žádost o spojení dynamicky vytváří individuální obslužný proces ,
- pakety mezi servrem a obslužnými procesy jsou posílány přímo (tj. bez vzájemného propojení těchto modulů)



Obsah souboru celek. ned:

simple Generator // generátor klientů (žádostí o spojení)
gates: **in**: in; **out**: out;
endsimple

simple Server // rodičovský modul generující Obslužné procesy
parameters doba_obsluhy: **numeric**; **gates** **in:** in; **out:** out;
endsimple

simple Obsluzny_proces // dynamicky generovaný modul
gates in: in; // nepřipojený vstup pro přímé zasílání zpráv
endsimple

module Testserver
submodules: gen: Generator; ser: Server;
connections: gen.out --> ser.in; gen.in <-- ser.out;
endmodule
network test :Testserver **endnetwork**

Příklad: generování obslužných procesů

Obsah souboru celek.msg:

- vnitřní struktura přenášených paketů:

```
message Paket
    {
       fields:  int zdr_adr;    // adresa klienta nebo servru
                int cil_adr;      // adresa klienta nebo servru
                int ident_obsluhy; // identifikace obslužného procesu
                double data [ 4 ] ;  //
    }
```

Typy paketů

- zde rozlišeny standardním atributem kind:
  - kind = 0 (žádost klienta o spojení),
  - kind = 1 (potvrzení spojení - od obslužného procesu),
  - kind = 2 (žádost klienta o zrušení spojení),
  - kind = 3 (potvrzení zrušení – od obslužného procesu),
  - kind = 4 (data – od klienta nebo obslužného procesu),

Příklad: generování obslužných procesů

Obsah souboru celek.cpp

```
# include “celek_m.h”
# include <omnetpp.h>
class Server : public cSimpleModule
 {  public:
    cModuleType * gen_modulu;  // gen.  modulů
    virtual void initialize ( );
    virtual void handleMessage ( cMessage* msg );
    Module_Class_Members ( Server, cSimpleModel, 0);
 } ;
class Obsluzny_proces : public cSimpleModule
 { public :
   virtual void activity ( ) ;
   Module_Class_Members (Obsluzny_proces,  cSimpleModel,
                                                    16384);
 } ;
class Generator : public cSimpleModule
{   protected:
    virtual void activity(); // udalostne orientovany popis
    Module_Class_Members (Generator, cSimpleModule, 16384 );
};
Define_Module (Server)
Define_Module (Obsluzny_proces)
Define_Module (Generator)
```

Příklad: generování obslužných procesů

```
void Generator :: activity ( )
{
  Paket* pkt = new Paket ("p1");
  pkt -> setName ("REQ");
  pkt -> setKind (0);  // zadost o spojeni
  pkt -> setZdr_adr ( 1 );  // adresa klienta = 1
  send ( pkt, "out„ );        // odeslani zadosti o spojeni
  ev << "klient1 posila zadost o navazani spojeni" << endl;
  pkt = ( Paket* ) receive ( );          // cekej na potvrzeni od serveru
  if ( pkt -> kind() == 1)
    { ev << " potvrzeni pro klienta 1 "<< endl;
      int ident_obsluhy = pkt -> getIdent_obsluhy ();  // id. obs. proc.
      pkt -> setKind (4);
      pkt -> setData ("ABCD");  // data
      ev << "klient1 posila data"  << endl;
      pkt -> setZdr_adr ( 1 );  // adresa klienta = 1
      pkt -> setIdent_obsluhy ( ident_obsluhy );
      send ( pkt, "out„ );
      pkt = (Paket*) receive();  // klient1 čeká na výsledek
      if ( (pkt->kind( ) == 4) && (pkt->getCil_adr( ) == 1))
              ev << "klient1 obdrzel data " << pkt->getData()<<endl;  }
      // následuje simulace dalšího klienta
     ……..atd.
}
```

Příklad: generování obslužných procesů

```
void Server : initialize ( )
 {  // následuje vytvoření generátoru obslužných procesů
     gen_modulu = findModuleType ( “Obsluzny_proces“ );
 }

void Server : handleMessage ( cMessage * msg )
{  Paket * pkt = check_and_cast <Paket *> ( msg );  // přetypování
   if ( pkt  -> kind ( ) == 0 )                  // žádost o nové spojení
       {   // následuje vytvoření nového obsluhujícího procesu a
           // spuštění jeho aktivity
           cModule* pm = gen_modulu  ->
                                createScheduleInit ( “ obsluha ”, this );
           // následuje odeslání paketu obsluhujícímu procesu
           sendDirect (  pkt, 0.0, pm, “in” ); // odeslání pkt modulu m
       }
       else    // nejde o navázání nového spojení
            {   // následuje identifikace dříve vytvořeného obslužného
                // procesu ( z položky ident_obsluhy )
              int  obsluha_ident = pkt  -> getIdent_obsluhy ( );
              // následuje získání pointru na identifikovaný modul
              cModule * pm = simulation . module ( obsluha_ident );
              sendDirect ( pkt,  0.0, pm,  “ in“ ) ; // předání paketu
                                                  // obslužnému procesu
            }
}
```

Příklad: generování obslužných procesů

```
void Obsluzny_proces : activity ( )  // procesově orientovaný popis
{
 // následuje reakce na žádost o navázání spojení
 // získání parametru od rodičovského modulu ( Server )
 cPar& doba_obsluhy = parentModule( ) ->par ( ”doba_obsluhy” );

 // získání pointru na výstupní port ”out” rodičovského modulu:
 cGate* vystup_serveru = parentModule( )  ->gate ( ”out” );

  Paket * pkt = ( Paket *) receive ( ); čeká na žádost o spojení
  int klient_adr = pkt -- > getZdr_adr ( );      // čtení adresy zdroje
  int server_adr = pkt -- > getCil_adr ( );      // čtení adresy serveru

  // příprava potvrzení žádosti o navázání spojení a odeslání
  pkt -- > setName ( “ACK “) ;           // nastavení jména zprávy
  pkt -- > setKind ( 1 ) ;               // příznak potvrzení
  pkt -- > setZdr_adr ( server_adr ) ;  // nastavení adresy zdroje
  pkt -- > setCil_adr ( klient_adr ) ;     // nastavení adresy cíle
  pkt -- > setIdent_obsluhy ( id ( ) ) ;   // vložení identifikačního
                                           // čísla modulu

  // odeslání potvrzení o navázaném spojení
  sendDirect ( pkt, 0 , vystup_serveru );
```

Příklad: generování obslužných procesů

```
// následuje zpracování dalších ( datových ) paketů
  while ( 1)
        {  pkt = ( Paket* ) receive ( );   // čekání na další paket
           int typ  = pkt -> kind ( );       // čtení typu paketu
             if ( typ  == 2 )   break ;       // žádost o zrušení spojení
             if ( typ  == 4 )                  // datový paket
                        {    // zpracování paketu a příprava odpovědi
                             ……..;  // nastavení položek paketu
                             wait ( doba_obsluhy );  // simulace doby
                             sendDirect ( pkt, 0 , vystup_serveru );
                        }
                        else   error ( " chyba v protokolu " );
        }

  // následuje příprava potvrzení o zrušení spojení
 pkt -> setKind ( 3 );   // příznak potvrzení o zrušení spojení
 ……..;            // nastavení ostatních položek
 sendDirect ( pkt, 0 , vystup_serveru );   // odeslání potvrzení
 deleteModule ( ) ;  // zrušení obsluhujícího modulu
}
```

Příklad: generování obslužných procesů

```
// následují naprogramované výstupy včetně informací o událostech:
** Event #0.  T=0.0000000  ( 0.00s).  Module #2 `dyn.gen'
klient1 posila zadost o navazani spojeni
** Event #1.  T=0.0000000  ( 0.00s).  Module #3 `dyn.ser'
prisla zadost o spojeni
** Event #2.  T=0.0000000  ( 0.00s).  Module #4 `dyn.ser.obsluha'
bezi activity obsl. procesu
** Event #3.  T=0.0000000  ( 0.00s).  Module #4 `dyn.ser.obsluha'
odesilam potvrzeni: 4
** Event #4.  T=0.0000000  ( 0.00s).  Module #2 `dyn.gen'
 potvrzeni prislo
klient1 posila data
** Event #5.  T=0.0000000  ( 0.00s).  Module #3 `dyn.ser'
** Event #6.  T=0.0000000  ( 0.00s).  Module #4 `dyn.ser.obsluha'
** Event #7.  T=       2  ( 2.00s).  Module #4 `dyn.ser.obsluha'
obsluha:4 zpracovavam data ABCD
** Event #8.  T=       2  ( 2.00s).  Module #2 `dyn.gen'
klient1 obdrzel data po zpracovaniAACC
// následují  pouze naprogramované výstupy (ostatní potlačeny):
klient2 posila zadost o navazani spojeni
prisla zadost o spojeni
bezi activity obsl. procesu
odesilam potvrzeni: 5
klient2 posila data
obsluha:5 zpracovavam data EFGH
klient2 obdrzel data po zpracovaniAACC
```

Získávání statistik

- objekty pro uchování statistik (instance tříd odvozených od základní třídy cStdDev):
  - konstruktor třídy: cStdDev :: cStdDev (const char* jméno)
  - příklad deklarace:

    cStdDev stat1 (" vysledky") ; // deklarace objektu stat1

- funkce pro shromažďování statistik:
  - virtual void cStdDev :: collect (double hodnota)
  - příklad:

    stat1. collect (h1); // modifikace objektu hodnotou h1

- metody třídy cSTdDev pro získání výsledných statistik:
  - virtual long cStdDev :: samples ( )…..vrací celkový počet sledovaných hodnot,
  - virtual double cStdDev :: min ()…..vrací minimální hodnotu,
  - virtual double cStdDev :: max ()…vrací maximální hodnotu,
  - virtual double cStdDev :: mean ( )…..vrací střední hodnotu,
  - virtual double cStdDev :: stddev ( )….vrací stand. odchylku,
  - virtual double cStdDev :: sum ( )……. vrací celkový součet

  - příklady:

    long počet_vzorku = stat1 . samples ( );
    double min = stat1 . min ( );
    double max = stat1. max ( );

Histogramy

• několik typů: odvozených od třídy cDensityEstbase,

Ekvidistantní rozložení mezí buněk:

– **třída cLongHistogram**:

- pro celočíselné šířky buněk (zachovává počet buněk a případně upravuje rozsah),
- konstruktor: cLongHistogram :: cLongHistogram
  (const char* jmeno, int počet_bunek =-1)

– **třída cDoubleHistogram**:

- pro reálné šířky buněk (zachovává rozsah a případně upravuje šířku buňky),
- konstruktor: cDoubleHistogram :: cDoubleHistogram
  (const char* jmeno, int počet_bunek =-1)

Stanovení počtu buněk: konstruktorem nebo funkcí

void cHistogramBase :: setNumCells (int počet)
……specifikuje počet buněk histogramu;

–cLongHistogram:

- není-li tento počet specifikován, pak se pro každé číslo x z rozsahu založí jedna buňka s mezemi x - 0.5 a x + 0.5 (max 10 000),
- je-li specifikován rozsah i počet, tak se rozsah upraví tak, aby byl celočíselným násobkem počtu buněk,

– cDoubleHistogram

- počet buněk je určen specifikací nebo 10 (implicitní hodnota).

Specifikace rozsahu a mezí

Stanovení rozsahu : explicitní nebo automatické

- **explicitní stanovení rozsahu**:

```
virtual void cDensityEstBase :: setRange
                                    ( double dm , double hm )
```

Příklady:

```
cLongHistogram  h1;
h1. setRange ( 0, 99 );
h1. setNumCells (7);
// zůstane zachován počet buněk,  jejich šířka se vypočte tak,
// aby specifikovaný rozsah byl plně pokryt
// h1:  7 buněk šířky 15:   < -0.5, 14.5, 29.5,…….89.5, 104.5 >

cDoubleHistogram h2 ;
h2. setRange ( 0, 99 );
h2. setNumCells (7);
// šířka buněk se přizpůsobí  specifikovanému rozsahu
// h2:  7 buněk šířky 14.1429  ( = 99 / 7)

cDoubleHistogram *  h3;
 h3 = new cDoubleHistogram  ( " h3 ", 40 );  // 40 buněk
 h3 -> setRange ( 0, 99 );
 // h3:  40 buněk šíře 2.475 ( = 99 / 40)
```

Histogramy

Nerovnoměrné rozložení mezí buněk: **třída cVarHistogram**

konstruktor: cVarHistogram :: cVarHistogram

(const char* jmeno, int počet_bunek, mod)

Stanovení mezí buněk:

- manuálně uživatelem
  - mod :: = HIST_TR_NO_TRANSFORM : manuální mód pro explicit. zadání mezí ,
  - 2. parametr v konstruktoru v tomto případě nemá význam,
  - addBinBound( ) … funkce pro zadávání mezí

- automaticky po dosažení určitého počtu hodnot tak aby počty v buňkách byly přibližně stejné,
  - mód :: = HIST_TR_AUTO_EPC_DBL: reálné šířky buněk
  - mód :: = HIST_TR_AUTO_EPC_INT: cel. šířky buněk
  - void cDensityEstBase :: setNumFirstVals (int počet)

// funkce specifikuje počáteční počet vzorků pro odhad
// mezí histogramu

Příklad:

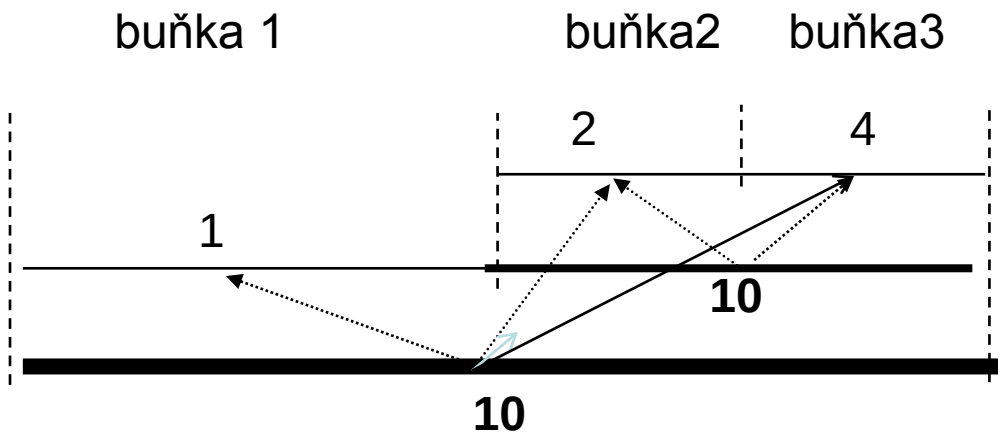
```
cVarHistogram h ( “hist “, 100,
                  HIST_TR_AUTO_ EPC_DBL );
// následuje stanovení mezí po obdržení 5000 vzorků
setNumFirstvals ( 5000 ) ;
```

Histogramy

Automatické dělení buněk: **třída cKSplitHistogram**

- začíná s rovnoměrným rozložením mezí buněk ve specifikovaném rozsahu,
- po dosažení určitého počtu hodnot v buňce se každá buňka štěpí do dvou dceřinných buněk d1, d2 s nulovým počátečním obsahem; původní obsah mateřské buňky m zůstává zachován a bude při výpočtu výsledné statistiky rozdělen v určitém poměru mezi obě dceřinné buňky,
- takto štěpení pokračuje do hloubky, dceřinný majetek dědí vnučky, atd.
- funkce . setNumCells (…) použít nelze.

Příklad:



Výsledné počty v buňkách: při stejných dědických právech

buňka1: 1 + 10/2 = 6

buňka 2: 2 + (10/2 + 10) /2 = 9.5

buňka3: 4 + (10/2 + 10) /2 = 11.5

Celkem = 27

Zápis vzorků, čtení a záznam histogramů

- korekce histogramu (zápis vzorku) :

  void cStdDev :: collect (double hodnota)

  příklad:

  cDoubleHistogram histo1 (“histogram “);

  histo1 . collect (1.5);

- zápis histogramu do souboru:

  void cLongHistogram :: saveToFile (FILE *f) const

  příklad:

  cLongHistogram histo2 (“ do souboru “);

  FILE * f2 = fopen (“ histogram2.dat“ , “w“); // otevř. pro zápis

  histo2. saveToFile (f2); // zápis objektu histo2 do souboru f2

  fclose (f2);

- čtení histogramu ze souboru:

  void cLongHistogram :: loadFromFile (FILE *f)

  příklad:

  cLongHistogram histo3 (“ ze souboru “);

  FILE * f2 = fopen (“ histogram2.dat“ , “r“); // otevření pro čtení

  histo3 . loadFromFile (f2); // naplnění objektu histo3 ze
  // souboru f2

  fclose (f2);

Získávání dat z histogramů

- následující funkce jsou definovány pro všechny typy histogramů:

virtual int cHistogramBase :: cells ( ) const = 0
…vrací počet buněk histogramu

virtual int cDensityEstBase :: cell (int k) = 0
… vrací počet vzorků v buňce k,

virtual double cDensityEstBase :: basepoint (int k) = 0
...vrací dolní mez k –té buňky

virtual double cDensityEstBase :: cellPDF (int k) const
...vrací pravděpodobnost p_k příslušející k-té buňce; tato pravděpodobnost je rovna relativní četnosti hodnot v této buňce a je vztažená na jednotku její šířky: $p_k = n_k / (N * s_k)$,

n_k ….počet hodnot v k-té buňce,

N…..celkový počet hodnot v histogramu,

s_k ….šířka k-té buňky

virtual int cDensityEstBase :: underflowCell ( )
…vrací počet vzorků pod dolní mezí,

virtual int cDensityEstBase :: overflowCell ( )
…..vrací počet vzorků nad horní mezí,

Získávání dat z histogramů

virtual long cStdDev :: samples ( )

….vrací celkový počet vzorků nashromážděných v histogramu,

virtual double cStdDev :: random ( )

…vrací náhodnou hodnotu jejíž rozložení je generováno dle daného histogramu,

Příklady:

```
double  sire_bunky, pdf ;
int   pocet;
cLongHistogram  histo ( “histogram “ );
……..;
long n          = histo . samples ( ) ;   // celkový počet vzorků
sire_bunky  = histo . basepoint (1) – histo . basepoint ( 0 );
                                          // sirka nulté buňky
pocet  = histo . cell ( 1 ) ;   // počet vzorků v buňce 1
// následuje výpočet pravděpodobnosti příslušející  k-té buňce
pdf      = histo . cellPDF ( k ) * ( histo . basepoint ( k+1 ) –
                                    histo . basepoint ( k+1 )  ) ;
```

Poznámka: empiricky získané rozložení náhodných hodnot produkovaných jedním modelem lze použít pro generování vstupů jiného modulu

```
double rnd = histo . random ( );
// příkaz generuje náhodné číslo dle rozložení v histo
```

Testy histogramů

Plnění šesti různých histogramů:

```
for  (  int  i = 1;  i < 101 ;  ++ i  )     // každá hodnota 100 krát
 {
    for ( int  j = 1;  j < 5 ; ++ j )        // uložení hodnot 1, 2, 3, 4
           {   histo1. collect ( j );  histo2. collect ( j );
               histo3->collect ( j ); histo4->collect ( j );
               histo5->collect ( j ); histo6->collect ( j )            }

  for ( int  j = 7;  j < 12 ; ++ j )       // uložení hodnot 7, 8, 9, 10, 11
           {  histo1. collect ( j );   histo2. collect ( j );
              histo3->collect ( j );  histo4->collect ( j );
              histo5->collect ( j );  histo6->collect ( j );            }
   }
```

Tisk výstupů: obvykle ve funkci finish ()

```
double pravd = 0.0;
for ( int i = 0; i < histo1 . cells (); ++i )
{   ev << "i = " << i << " cell  (i)= " << histo1.cell (i) <<
    " hustota_pr =  " << histo1.cellPDF(i)* ( histo1.basepoint (i+1) -  histo1.
     basepoint(i) ) << " dolni mez =   " << histo1. basepoint(i) << endl;
     pravd = pravd + histo1. cellPDF (i)* ( histo1.basepoint (i+1) -
                histo1. basepoint(i) );
}
ev << "celkem =   " << histo1. samples()  << " soucet pravdepodobnosti =
                              " << pravd << endl;
ev << "underflow =   " << histo1. underflowCell() << " overflow = "
                                                                    47
                              << histo1. overflowCell() << endl;
```

Výsledky testů

Nastavení histogramu:

```
cLongHistogram histo1;
histo1. setRange ( 0,15 );    // musi byt zadano
histo1. setNumCells ( 10 );  //  sire bunky  = 2, upraven rozsah
```

Test histogramu histo1:

```
i = 0 cell  (i)= 100    hustota_pr =  0.111111    dolni mez =   -0.5
i = 1 cell  (i)= 200    hustota_pr =  0.222222    dolni mez =   1.5
i = 2 cell  (i)= 100    hustota_pr =  0.111111    dolni mez =   3.5
i = 3 cell  (i)= 100    hustota_pr =  0.111111    dolni mez =   5.5
i = 4 cell  (i)= 200    hustota_pr =  0.222222    dolni mez =   7.5
i = 5 cell  (i)= 200    hustota_pr =  0.222222    dolni mez =   9.5
i = 6 cell  (i)= 0      hustota_pr =  0           dolni mez =   11.5
i = 7 cell  (i)= 0      hustota_pr =  0           dolni mez =   13.5
i = 8 cell  (i)= 0      hustota_pr =  0           dolni mez =   15.5
i = 9 cell  (i)= 0      hustota_pr =  0           dolni mez =   17.5

celkem =   900          soucet pravdepodobnosti = 1
underflow =   0         overflow = 0
```

Poznámka: buňka obsahuje dolní mez

Výsledky testů

Nastavení histogramu:

```
cDoubleHistogram histo2;
histo2 . setRange ( 0,15 );   // musi byt zadano
histo2 . setNumCells ( 10 ); // sire bunky = 1.5, rozsah zůstává 15
```

Test histogramu histo2:

```
i = 0 cell  (i)= 100   hustota_pr =  0.111111  dolni mez: 0      // 1
i = 1 cell  (i)= 100   hustota_pr =  0.111111  dolni mez: 1.5    // 2
i = 2 cell  (i)= 200   hustota_pr =  0.222222  dolni mez: 3      // 3, 4
i = 3 cell  (i)= 0     hustota_pr =  0         dolni mez: 4.5
i = 4 cell  (i)= 100   hustota_pr =  0.111111  dolni mez: 6      // 7
i = 5 cell  (i)= 100   hustota_pr =  0.111111  dolni mez: 7.5    // 8
i = 6 cell  (i)= 200   hustota_pr =  0.222222  dolni mez: 9      // 9,10
i = 7 cell  (i)= 100   hustota_pr =  0.111111  dolni mez: 10.5 // 11
i = 8 cell  (i)= 0     hustota_pr =  0         dolni mez: 12
i = 9 cell  (i)= 0     hustota_pr =  0         dolni mez: 13.5
celkem =   900      soucet pravdepodobnosti = 1
underflow =   0      overflow = 0
```

Poznámka: buňka obsahuje vždy dolní mez

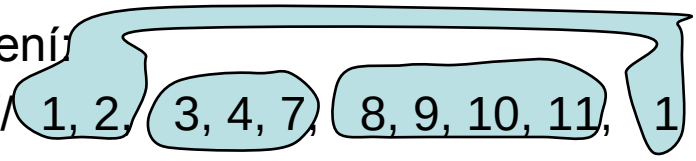
Test histogramů

Nastavení histogramu:

```
cVarHistogram* histo3;
histo3 = new cVarHistogram (“hist", 3 ,
                                    HIST_TR_AUTO_EPC_DBL);
// automatické vytvoření mezí  tří buněk po shromáždění
// zadaného počtu vzorků ( ve snaze dosáhnout vyrovnaných
// četností v buňkách)
// výstup při následujicím nastavení:
histo3->setNumFirstVals ( 9 );    // počet pro vytvoření mezí

i = 0 cell  (i)= 300  dolni mez: 1   hustota_pr =  0.333333 // 1,2,3
i = 1 cell  (i)= 300  dolni mez: 4   hustota_pr =  0.333333 // 4,7,8
i = 2 cell  (i)= 300  dolni mez: 9   hustota_pr =  0.333333 // 9,10,11
      soucet pravdepodobnosti = 1
celkem =   900

// výstup při následujícím nastavení:
histo3->setNumFirstVals ( 10 ); //
```




```
i = 0 cell  (i)= 200  dolni mez: 1 hustota_pr =  0.222222 //1,2,
i = 1 cell  (i)= 300  dolni mez: 3 hustota_pr =  0.333333 // 3,4,7
i = 2 cell  (i)= 400  dolni mez: 8 hustota_pr =  0.444444 //8,9,10,11
      soucet pravdepodobnosti = 1
celkem =   900
```

Test histogramů

Nastavení histogramu:

```
cVarHistogram* histo4;
histo4 = new cVarHistogram ( " hist",
                                    11, HIST_TR_NO_TRANSFORM);
// následuje explicitní stanovení mezí
histo4->addBinBound (1.0 );
histo4->addBinBound (2.);
histo4->addBinBound (3.);
histo4->addBinBound (4.);
histo4->addBinBound (5.);
histo4->setRange (1,15);  // v případě módu NO TRANSFORM
                          //  musí být definován rozsah
```

Test histogramu histo4:

```
i = 0 cell  (i)= 100  dolni mez: 1     hustota_pr =  0.111111
i = 1 cell  (i)= 100  dolni mez: 2     hustota_pr =  0.111111
i = 2 cell  (i)= 100  dolni mez: 3     hustota_pr =  0.111111
i = 3 cell  (i)= 100  dolni mez: 4     hustota_pr =  0.111111
i = 4 cell  (i)= 500  dolni mez: 5     hustota_pr =  0.555556
      soucet pravdepodobnosti = 1
celkem =   900
underflow =   0       overflow = 0
```

Test histogramů

Nastavení histogramu:

cKSplit *histo5; histo5 = new cKSplit; histo5->setRange (0,15);

Demonstrace štěpení buněk histogramu histo5:

- situace po čtyřech hodnotách 1,2,3,4: (pův. rozsah rozpůlen) :

i = 0 cell (i)= 4 dolni mez: 0 hustota_pr = 1

i = 1 cell (i)= 0 dolni mez: 7.5 hustota_pr = 0

soucet pravdepodobnosti = 1

- rozpůlení dolní buňky po pěti hodnotách 1,2,3,4,1:

i = 0 cell (i)= 4 dolni mez: 0 hustota_pr = 0.8

i = 1 cell (i)= 1 dolni mez: 3.75 hustota_pr = 0.2

i = 2 cell (i)= 0 dolni mez: 7.5 hustota_pr = 0

- rozpůlení dolní buňky po pěti hodnotách 1,2,3,4,4:

i = 0 cell (i)= 1 dolni mez: 0 hustota_pr = 0.2

i = 1 cell (i)= 4 dolni mez: 3.75 hustota_pr = 0.8

i = 2 cell (i)= 0 dolni mez: 7.5 hustota_pr = 0

Poznámka:

- podinterval štěpené buňky, do něhož padne hodnota která způsobila štěpení, se stává dceřinou buňkou „dědící 75% jmění od své mateřské buňky“;
- zbývající dceřinná buňka dědí 25 %.

Test histogramů

- situace po devíti stech původních hodnotách:

i = 0 cell (i)= 5.06078 dolni mez: 0 hustota= 0.00562309
i = 1 cell (i)= 83.3867 dolni mez: 0.9375 hustota = 0.0926519 // 1
i = 2 cell (i)= 11.7956 dolni mez: 1.40625 hustota = 0.0131062
i = 3 cell (i)= 88.2097 dolni mez: 1.875 hustota = 0.0980107 // 2
i = 4 cell (i)= 10.7366 dolni mez: 2.34375 hustota = 0.0119295
i = 5 cell (i)= 89.2562 dolni mez: 2.8125 hustota = 0.0991735 // 3
i = 6 cell (i)= 10.7521 dolni mez: 3.28125 hustota = 0.0119467
i = 7 cell (i)= 82.7497 dolni mez: 3.75 hustota = 0.0919441 // 4
i = 8 cell (i)= 11.9166 dolni mez: 4.21875 hustota = 0.0132406
i = 9 cell (i)= 5.22209 dolni mez: 4.6875 hustota = 0.00580233
i = 10 cell (i)= 5.22851 dolni mez: 5.625 hustota = 0.00580946
i = 11 cell (i)= 83.7641 dolni mez: 6.5625 hustota = 0.0930713 // 7
i = 12 cell (i)= 11.9214 dolni mez: 7.03125 hustota = 0.013246
i = 13 cell (i)= 10.6875 dolni mez: 7.5 hustota = 0.011875
i = 14 cell (i)= 89.0625 dolni mez: 7.96875 hustota = 0.0989583 // 8
i = 15 cell (i)= 10.6875 dolni mez: 8.4375 hustota = 0.011875
i = 16 cell (i)= 89.0625 dolni mez: 8.90625 hustota = 0.0989583 // 9
i = 17 cell (i)= 10.6875 dolni mez: 9.375 hustota = 0.011875
i = 18 cell (i)= 89.0625 dolni mez: 9.84375 hustota = 0.0989583 // 10
i = 19 cell (i)= 10.6875 dolni mez: 10.3125 hustota = 0.011875
i = 20 cell (i)= 89.0625 dolni mez: 10.7813 hustota = 0.0989583 // 11
i = 21 cell (i)= 1 dolni mez: 11.25 hustota = 0.001111
soucet pravdepodobnosti = 1
celkem = 900

Test histogramů

Nastavení histogramu:

```
cPSquare *histo6;
histo6 = new cPSquare;     // bunky se stejnou pravdepodobnosti
```

Test histogramu histo6:

```
i = 0 cell  (i)= 91  dolni mez: 1 ???        hustota_pr =  0
i = 1 cell  (i)= 89  dolni mez: 1 ???        hustota_pr =  0.0988889
i = 2 cell  (i)= 90  dolni mez: 2.00008      hustota_pr =  0.1
i = 3 cell  (i)= 90  dolni mez: 3.00524      hustota_pr =  0.1
i = 4 cell  (i)= 90  dolni mez: 4.02233      hustota_pr =  0.1
i = 5 cell  (i)= 90  dolni mez: 7.0313       hustota_pr =  0.1
i = 6 cell  (i)= 90  dolni mez: 8.01062      hustota_pr =  0.1
i = 7 cell  (i)= 90  dolni mez: 9.00259      hustota_pr =  0.1
i = 8 cell  (i)= 90  dolni mez: 9.7148       hustota_pr =  0.1
i = 9 cell  (i)= 90  dolni mez: 10.2515      hustota_pr =  0.1

    soucet pravdepodobnosti = 0.898889    ??????
    celkem =   900
    underflow =   0       overflow = 0
```

Podporované typy výstupů

1) Vektory (hodnoty s časovými známkami): ukládány do souboru pomocí objektů třídy cOutVect:

   – jméno souboru lze specifikovat v inicializačním souboru omnet.ini
   – implicitně do souboru omnetpp.vec,
   – soubor vektorů je rušen na počátku každého simulačního běhu,
   – ukládání vektorů lze v průběhu simulace střídavě blokovat či uvolňovat (v programu i v konfiguračním souboru),

   virtual bool cOutVector :: record (double val)
   – provede zápis hodnoty val spolu s okamžitou hodnotou modelového času do specifikovaného souboru,
   – vrací true, pokud hodnota val byla skutečně zapsána, jinak false (v případě neuvolněného objektu),

Příklad: :

```
cOutVector delka_fronty;
        delka_fronty . record ( fr. length ( ) ) ;

//  formát řádku v souboru omnetpp.vec:
  <identifikační číslo vektoru> <hodnota času> <hodnota val>
```

Podporované typy výstupů

virtual bool cOutVector :: recordWithTimestamp
(simtime_t t , double val)

- provede zápis hodnoty val spolu s hodnotou t časové známky do specifikovaného souboru,
- vrací true, pokud došlo k zápisu, jinak false (v případě neuvolněného objektu),
- časové známky jednotlivých volání musí tvořit neklesající posloupnost,

virtual void cOutVector :: enable ( )

- umožní zápis hodnot metodou record (objekt je implicitně uvolněn),

virtual void cOutVector :: disable ( )

- znemožní zápis hodnot metodou record ; případné použití této metody vrací hodnotu false bez jakéhokoliv efektu,

virtual long cOutVector :: valuesStored ( )

- vrací skutečný počet hodnot zapsaných prostřednictvím daného objektu,
- neuvažuje hodnoty v době kdy objekt nebyl uvolněn,

virtual long cOutVector :: valuesReceived ( )

- vrací celkový počet hodnot předaných danému objektu,

Podporované typy výstupů

2) Skaláry:

- jednotlivé shrnující hodnoty charakterizující simulační běh (střední hodnota, atd.)
- ukládány implicitně do souboru omnetpp.sca obvykle na konci simulačního běhu (při spuštění funkce finish( )),
- mezi jednotlivými běhy se neruší, ale hodnoty se připojují za hodnoty z minulého běhu),
- vlastní zápis hodnot: pomocí funkce recordScalar ( ) různých tříd:

void cSimpleModule :: recordScalar
(const char* jmeno, double val)

- provede zápis textu a hodnoty val do souboru,

Příklad:

```
double prumerny_tok = počet_prenesenych_bitu / simTime ( )
recordScalar ( " Průměrný tok = ", prumerny_tok );
```

virtual void cStatistic::recordScalar (const char * jmeno = NULL)

- provede záznam veškerých statistik daného objektu (mean, max,…) pomocí několikaterého volání funkcí recordScalar třídy cSimpleModule,

Příklad:

```
cDoubleHistogram * histo;
histo → recordScalar ( " Histogram" );  // metoda histogramu
```

Příklad: SHO: viz dříve (včetně statistik)

```
class Likvidator : public cSimpleModule       // typ Likvidator
{   public:
    cStdDev statistika_dob;     // deklarace objektu pro statistiku
    cDoubleHistogram *histo1;  // histogram  čekacích dob
    virtual void initialize( );
    virtual void handleMessage (cMessage * msg);
    virtual void finish( );
    Module_Class_ Members ( Likvidator, cSimpleModule, 0 )  };

class Generator : public cSimpleModule
{ public: virtual void activity( );   // procesove orientovany popis
  Module_Class_ Members (Generator, cSimpleModule,16384)  };

class Kanal : public cSimpleModule
{  protected:
    cMessage *konec_obsluhy, *monitor; // vlastní zpráva+monitor
    cQueue fr;                          // deklarace fronty
    cLongHisto * histo2;               // ptr na histogram délky fronty
    cOutVector delka_fronty ( " V1" ) ; // okamžitá délka + čas
    virtual void initialize();
    virtual void handleMessage(cMessage * msg );
    virtual void finish();
    Module_Class_ Members ( Kanal, cSimpleModule, 0 )    };
```

Příklad SHO

```
void Generator::activity( )                    // beze změny
{   int  pocet_poz  = par("pocet_uloh");    // parametry z NED
    cPar& interval  = par ("interval");
    for ( int i=0; i < pocet_poz ; i++ )        // generovani požadavků
     { char oznaceni_poz [32];
        sprintf ( oznaceni_poz, „pozadavek-%d", i );  // cislovani poz
        cMessage *poz = new cMessage ( oznaceni_poz );
        poz->setTimestamp( ); // nastaveni casove znamky požad.
        send( poz, "out" );          // odesláni požadavku pres port out
        wait ( (double) interval ); // interval mezi prichody uloh      }
   endSimulation ( );  // nasilne ukonci (kvuli vzorkovani fronty)
}
void Likvidator::initialize()
{    statistika_dob . setName ("Statistika celkovych dob");
     histo1. setRange (0, 30);    // rozsah pro histogram dob
     int pbunek = par ( “pocet_bunek “);
     histo1 . setNumCells ( 15 );    }
void Likvidator::handleMessage(cMessage *poz)
{   // následuje shromažďování statistik v průběhu simulace
    double d = simTime() - poz->timestamp( );
 // celkova doba požadavku strávená ve frontě a obsluze
    statistika_dob . collect ( d );  // korekce statistiky
    histo1 . collect ( d );             // korekce histogramu
    delete poz;    }                    // likvidace požadavků
```

Příklad SHO

```
void Likvidator :: finish ( )
{
   ev << "*** Module: " << fullPath() << "***" << endl;
   ev << “Obslouzeno: " << statistika_dob . samples( ) <<  endl;
   ev << "prumer_doba:    " << statistika_dob . mean() << endl;
   ev << "min_doba:    " << statistika_dob . min() << endl;
   ev << "max_doba:    " << statistika_dob . max() << endl;
   ev << „standardni_odchylka:  " << statistika_dob . stddev( )
            << endl;
   double pravd = 0.0;
   for ( int i = 0; i < histo1 . cells (); i++ )
      { ev << "i = " << i << " cell  ( i ) = " << histo1. cell (i)<<
        "dolni mez: " << histo1. basepoint(i) <<
        "hustota_pr =  " << histo1. cellPDF(i) *
         ( histo1. basepoint (i+1) - histo1. basepoint(i) ) << endl;
        pravd = pravd + histo1 . cellPDF (i)*
              ( histo1 . basepoint(i+1)  - histo1 . basepoint (i) );
      }
   ev << "       soucet pravdepodobnosti = " << pravd << endl;
   ev << "celkem =   " << histo1.samples()  << endl;
   ev << "underflow =   " << histo1.underflowCell() << "
            overflow = " << histo1.overflowCell() << endl;
  // následuje záznam histogramu “histo“ do omnetpp.sca
   histo1->recordScalar ("vysledne statistiky");
}
```

Příklad SHO

```
void Kanal::initialize()
{
   fr . setName ( "fronta“ );          // nastaveni jména objektu fr
   int pbunek = par ( "pocet_bunek„ );
   // následuje generovani histogramu délky fronty a jeho
   // nastaveni
   histo2 = new cLongHistogram („ delka  fronty");
   histo2 -> setRange ( 0, 15 );
   delka_fronty . setName  ("okamzity pocet uloh v sho");
   //  následuje generování vlastní zpravy pro indikaci konce
   //  obsluhy ( časování obsluhy )
   konec_obsluhy = new cMessage ( „obslouzeno„ );
   monitor = new cMessage ( “ monitor” );
   scheduleAt ( simTime ( ) + 1.0,  monitor );
}


void Kanal :: finish ();
{
 /*  zde má být výstup histogramu histo2, který zaznamenává
    délky fronty :  analogie výstupu histo1 z funkce Likvidator ::
    finish ()
*/
}
```

Příklad SHO

```
void Kanal :: handleMessage (cMessage *msg)   // příjem zprávy
{  if ( ! msg == isSelfMessage( ) )     // příchod nového požadavku
    { if ( fr . empty ( ) )                    // prazdny kanal obsluhy
          {  delka_fronty . record ( 1 );  // zapis vektoru
             fr . insert ( msg );
             simtime_t  doba_obsluhy =par ( “doba_obsluhy");
             scheduleAt
                    ( simTime() + doba_obsluhy, konec_obsluhy );
          }  else   {   fr . insert ( msg );
                        delka_fronty . record ( fr->length ( ));   }
    }  else     // příjem zprávy o ukončení obsluhy
        {    if ( msg == konec_obsluhy )
             {  cMessage * obslouzeny = ( cMessage * ) fr . pop();
                send ( obslouzeny,  “ out“ );   // odchod požadavku
                delka_fronty . record ( fr->length ( ) );  // vektor
                if ( ! fr . empty ( ) )     // další do obsluhy
                  { simtime_t doba_obsluhy =par ( “doba_obsluhy");
                    scheduleAt
                      (simTime( ) + doba_obsluhy, konec_obsluhy );}
             }
            if  ( msg == monitor )
                 {   histo2 -> collect ( fr . length ( ) );
                     scheduleAt ( simTime ( ) + 1.0, monitor );  }
        }
}
```

Příklad SHO: výsledné histogramy

Histogram histo1 celkové doby strávené v systému:

- počet buněk = 15, rozsah = < 0, 30 >

i = 0 cell (i)= 331021 dolni mez: 0 hustota_pr = 0.331021
i = 1 cell (i)= 220546 dolni mez: 2 hustota_pr = 0.220546
i = 2 cell (i)= 147383 dolni mez: 4 hustota_pr = 0.147383
i = 3 cell (i)= 99291 dolni mez: 6 hustota_pr = 0.099291
i = 4 cell (i)= 65772 dolni mez: 8 hustota_pr = 0.065772
i = 5 cell (i)= 43348 dolni mez: 10 hustota_pr = 0.043348
i = 6 cell (i)= 29187 dolni mez: 12 hustota_pr = 0.029187
i = 7 cell (i)= 19401 dolni mez: 14 hustota_pr = 0.019401
i = 8 cell (i)= 12939 dolni mez: 16 hustota_pr = 0.012939
i = 9 cell (i)= 9329 dolni mez: 18 hustota_pr = 0.009329
i = 10 cell (i)= 6745 dolni mez: 20 hustota_pr = 0.006745
i = 11 cell (i)= 4801 dolni mez: 22 hustota_pr = 0.004801
i = 12 cell (i)= 3178 dolni mez: 24 hustota_pr = 0.003178
i = 13 cell (i)= 2127 dolni mez: 26 hustota_pr = 0.002127
i = 14 cell (i)= 1548 dolni mez: 28 hustota_pr = 0.001548

soucet pravdepodobnosti = 0.996616
celkem = 1000000
underflow = 0 overflow = 3384

prumer_doba: 5.05069
max. doba: 58.8296
min. doba: 8.80449e-006
standard. odchylka: 5.1857

Příklad SHO: výsledné histogramy

Histogram histo2 pro váženou délku fronty:

- počet buněk nebyl specifikován, rozsah = < 0, 15 >

i = 0 cell (i)= 249735 dolni mez: -0.5 hustota_pr = 0.199969
i = 1 cell (i)= 200774 dolni mez: 0.5 hustota_pr = 0.160765
i = 2 cell (i)= 161135 dolni mez: 1.5 hustota_pr = 0.129025
i = 3 cell (i)= 127607 dolni mez: 2.5 hustota_pr = 0.102178
i = 4 cell (i)= 101504 dolni mez: 3.5 hustota_pr = 0.0812767
i = 5 cell (i)= 81415 dolni mez: 4.5 hustota_pr = 0.0651909
i = 6 cell (i)= 64774 dolni mez: 5.5 hustota_pr = 0.0518661
i = 7 cell (i)= 51894 dolni mez: 6.5 hustota_pr = 0.0415528
i = 8 cell (i)= 41247 dolni mez: 7.5 hustota_pr = 0.0330275
i = 9 cell (i)= 32746 dolni mez: 8.5 hustota_pr = 0.0262205
i = 10 cell (i)= 25946 dolni mez: 9.5 hustota_pr = 0.0207756
i = 11 cell (i)= 21106 dolni mez: 10.5 hustota_pr = 0.0169001
i = 12 cell (i)= 16924 dolni mez: 11.5 hustota_pr = 0.0135515
i = 13 cell (i)= 13578 dolni mez: 12.5 hustota_pr = 0.0108722
i = 14 cell (i)= 10815 dolni mez: 13.5 hustota_pr = 0.00865983
i = 15 cell (i)= 8614 dolni mez: 14.5 hustota_pr = 0.00689744

soucet pravdepodobnosti = 0.968727
celkem vzorků fronty = 1248870
underflow = 0 overflow = 39056

prumer. delka fronty: 4.04399
standard. odchylka delky fronty: 4.64219
max. delka fronty: 60
min. delka fronty: 0

Generátory pseudo - náhodných čísel

- použití: při specifikaci aktuálních parametrů modulů, v popisu chování jednoduchých modulů,

Funkce pro generování rovnoměrného rozložení:

– long intrand (long n)… vrací hodnoty typu long z intervalu < 0, n -1>; používá nultý-tý generátor,

– long intrand (long n, int k)… vrací hodnoty typu long z intervalu < 0, n -1>; používá k-tý generátor,

– double dblrand ( )… vrací hodnoty typu double z intervalu < 0, 1); používá nultý-tý generátor,

– double dblrand (int k)… vrací hodnoty typu double z intervalu < 0, 1); používá k-tý generátor,

Transformační metody pro spojitá rozložení:

– double uniform (a, b, rng = 0)..vrací hodnoty typu double s rovnoměrným rozložením v intervalu < a, b),

– double exponential (s, rng = 0).. vrací hodnoty typu double s exponenciálním rozložením se střední hodnotou s,

Generátory pseudo - náhodných čísel

– double normal (s, b, rng = 0).. vrací hodnoty typu double s normálním rozložením se střední hodnotou s a směrodatnou odchylkou b ,

– double truncnormal (s, b, rng = 0).. vrací hodnoty typu double s normálním rozložením , které je omezeno na nezáporné hodnoty se střední hodnotou s a směrodatnou odchylkou b ,

– další : gamma, beta, chi_sguare, student, weitbull, triang, atd (viz manual) .

Transformační metody pro diskrétní rozložení:

– int intuniform (a, b, rng = 0)..vrací hodnoty typu int s rovnoměrným rozložením v intervalu < a, b)

– int bernoulli (p, rng = 0) .. vrací hodnotu 1, resp. 0 s pravděpodobností p, resp.1 - p

– int poisson (s, rng = 0).. vrací hodnoty typu int s Poissonovým rozložením se střední hodnotou s

– další: binomial, geometric, negbinomial (viz manual)

Generátory pseudo - náhodných čísel

- konfigurace generátorů - v souboru omnetpp.ini:
  - pro všechny běhy – v sekci [General],
  - pro jednotlivé simulační běhy – v sekci [Run n]
    - typ generátoru: rng-class = "cMersenneTwister" nebo "cLCG32" nebo "cAkaroaRNG"
    - celkový počet generátorů: num-rngs = …..
    - volba semen:
      - automatická: odvozena od čísla simulačního běhu a čísla generátoru,
      - pomocí programu seed tool (pouze pro cLCG32)
      - specifikovaná uživatelem (pro generátor 0):

        seed - 0 - lcg32 = 1082809519 // pro 0-tý generátor

- přístup k vlastnostem generátorů:
  - cRNG* cModule :: rng (int k) const … vrací ukazatel na k-tý generátor,
  - virtual unsigned long cRNG :: numbersDrawn ( )…vrací celkový počet dosud vygenerovaných čísel daným generátorem,
  - virtual unsigned int cRNG :: intRandMax ( )…vrací maximální možnou hodnotu daného generátoru,

Konfigurační soubor: omnetpp . ini

- účel: počáteční nastavení simulátoru,
  - libovolné pořadí sekcí,
  - komentáře: řádky začínající # nebo ;
- syntax souboru:
  - [General] ……nastavení pro všechny simulační běhy
  - [Run 1]…… …….nastavení 1. běhu…….
  - [Run n]…… ……nastavení n. běhu
  - [CMdenv] ……pro textové rozhraní
  - [Tkenv]……pro grafické rozhraní
  - [Parameters]……. hodnoty parametrů
  - [OutVectors]……. filtrace výstupních vektorů

<u>Položky sekce General :</u>

network = < jméno sítě >
snapshot -file = < jméno souboru >
output-vector-file = < jméno souboru >
output-scalar-file = < jméno souboru >
pause-in-sending = no | yes ,
pokud yes pak send = 2 kroky
sim-time-limit = < horní mez simulačního času >
cpu-time-limit = < horní mez reálného času >
num-rngs = < počet generátorů náhodných čísel>
další: zadání semen, celková velikost zásobníku, atd

Konfigurační soubor

Nastavení parametrů :

- v sekci [General] nebo [Run n] (při rozporu tyto mají přednost),
- parametry specifikované v NED nelze přepsat,
- identifikace parametrů: hierarchicky, př. celek. gen . pocet = …

- hromadná specifikace parametrů:
  * ….reprezentuje jeden nebo více znaků (kromě .),
  **…reprezentuje jeden nebo více znaků, příklady:
  sit . mod1. pocet = 10000; parametr pocet instance mod1,
  . mod. pocet =100; jde všechny objekty mod*,
  **. pocet = 100; jde o všechny parametry pocet,

- v případě vektorů modulů lze použít řezy:
  – příklady:
  **. m [2..4] . pocet = …# jde o instance m [2], m [3], m [4]
  **. m [.. 1] . pocet = …….# jde o instance m [0], m [1],
  **. m [7 ..] . pocet = …… # jde o instance m [7] a více
  **. m [*] . pocet = ……. # jde o zbývající m [5], m [6]

  # následuje nastavení všech parametrů na implicitní hodnoty
  # (pokud jsou parametry určeny funkcí input ( ) v NED);
  <spec. parametru>. use-default = yes
  # parametr nastaven dle input v NED (jinak 0, false, Null)

Konfigurační soubor

Filtrace výstupních vektorů :

- možnost: v sekcích [Run n] nebo [OutVectors]
- efekt: zapínání nebo vypínání záznamu jednotlivých objektů třídy cOutVector do souboru omnetpp.vec
- syntax:

  # následuje stanovení mezí časového intervalu ve kterém se
  # provádí zápis daného vektoru:

  <hierarchická specifikace modulu> .

  < jméno vektoru > . interval = <dolní mez> .. <horní mez>

  | .. <horní mez>

  | <dolní mez> ..

  # následuje zapnutí, resp. vypnutí funkce objektu:

  <hierarchická specifikace modulu> .

  < jméno vektoru> . enabled = yes | no

  # yes… zapnuto, no …vypnuto

Příklad:

předpoklad: existuje deklarace cOutVector vekt (“V1“);

** . V1. enabled = no # zablokování zápisu vektorů V1 všech
# modulů

vymezení intervalu pro zápis všech nezablokovaných

vektorů (bez ohledu na jejich jména) uvnitř všech modulů

** . interval = 30s ..100s

Pomůcky pro ladění

- **breakpoints:**
  - voláním funkce breakpoint (“ text“);
    - text umožní lokalizaci bodu přerušení,
    - použitelné pouze v procesu (tj. uvnitř funkce activity ()),

- **snapshots:**
  - snapshot (this); // výpis info o modulu: model. čas,
    // hodnoty atributů, odeslané zprávy,
  - snapshot (&simulation . msgQueue); // výpis obsahu FEL
    // včetně zdrojových a cílových modulů,
  - snapshot ( ); // výpis informací o všech modulech

    poznámka: snímky jsou ukládány seriově do souboru omnetpp . sna

Příklad:

```
snapshot ( this );
breakpoint (“ prave byl vytvoren snimek tohoto modulu “);
```

- **watches:**
  - volání makra WATCH (<identifikator>); // umožní zařadit specifikovanou proměnnou do seznamu veličin, jejichž hodnoty lze sledovat v průběhu trasování

Příklad:

```
int interval;
WATCH ( interval);
```