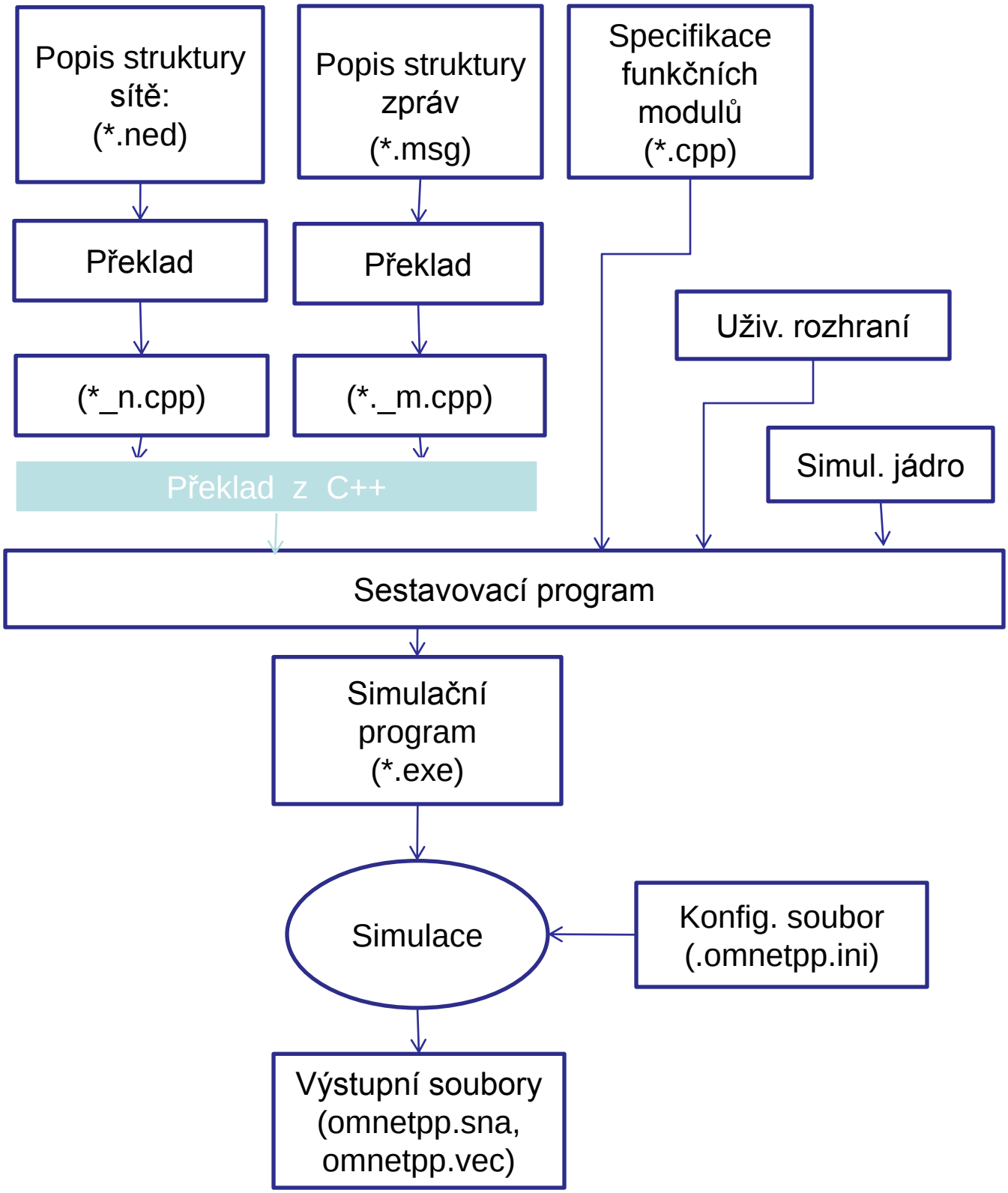
OMNET++

Literatura:

OMNET++ version 3.1, User Manual

www.omnetpp.org

Koncepce systému OMNET++



Charakteristika systému OMNET++

- pouze pro akademické instituce,
- komerční verse: OMNEST
- autor: Andreas Varga
- model:
  - network: definuje síť propojených instancí určitých typů dílčích modulů (submodules) v jazyku NED (network description),
  - typy dílčích modulů:
    - jednoduchý module (simple), definuje:
      - rozhraní (vstup. a výst. porty v jazyku NED),
      - parametry v jazyku NED,
      - chování (specifikace v C++ za podpory simulační knihovny),
    - strukturní modul (module), definuje:
      - rozhraní,
      - vytvoření instancí dílčích modulů,
      - vzájemné propojení dílčích modulů pomocí komunikačních kanálů,
  - komunikace mezi moduly:
    - pomocí přenosu zpráv definované struktury,
    - přímo do cílového modulu (bez přímého spojení),
    - s využitím nadefinovaných komunikačních kanálů

Základní filosofie systému OMNET++

• zpracování modelu: generování objektů pro reprezentaci modulů a jejich vzájemné propojení (dle souborů NED),

• inicializace modulů:

– provedení funkcí initialize ( ) všech událostně orientovaných modulů a spuštění funkcí activity ( ) procesově orientovaných modulů; inicializace složeného modulu vždy předchází inicializaci příslušných dílčích modulů,

– založení prvních událostí do FEL (future event list),

• řízení simulace dle obsahu FEL:

while (není konec simulace)

{

- vyjmi 1. událost z FEL,
- nastav simtime_t t = hodnota času právě vyjmuté události,
- proveď danou událost (handle Message příslušného modulu) ; důsledek: modifikace modelu + případné naplánování dalších událostí), }

• ukončení simulace:

– provedení funkcí finish ( ) všech modulů; pořadí: od funkčních modelů nahoru: určeno hierarchií modulů

Typy modulů

• jazyk NED (network description), komponenty jazyka:

– příkazy pro import použitých deklarací z jiných souborů:

• import " <jméno souboru>. ned",

– vytvoření typů jednoduchých modulů: rozhraní + parametry

simple < identifikátor typu modulu >

[**parameters:** …….;] // sekce parametrů

gates:……..; // sekce rozhraní

endsimple

– vytvoření typů složených modulů + specifikace struktury,

module < identifikátor typu modulu >

[**parameters:** ….;] // sekce parametrů

gates:……; // sekce vstupů a výstupů

submodules: ….; // sekce dílčích instancí

connections: ….; // sekce komunikačních kanálů

endmodule

– definice komunikačních kanálů:

channel **<** typ kanálu **>**

delay <zpoždění kanálu > ;

error <pravděpodobnost chyby jednoho bitu>;

datarate < přenosová kapacita kanálu > ;

endchannel

– vytvoření výsledné sítě:

network < ident. instance>: <identifikátor typu modulu>

endnetwork

Sekce parametrů modulu:

účel: specifikovat parametry modulu:

- jednoduché moduly (simple):
  - dimenze vstupů a výstupů,
  - parametry chování modulu,
- složené moduly:
  - dimenze vstupů a výstupů,
  - počty instancí dílčích modulů,
  - typy dílčích modulů,
  - inicializace parametrů dílčích modulů,

• syntax sekce parametrů:

parameters: < identifikátor par > : < typ par > ,……;

// zde pouhý výčet parametrů,

< typ par > : = **numeric, numeric const, const, bool, string**,

- **const**: lze pouze inicializovat (i náhodně),
- **numeric const:** lze pouze číst (vstup náh. posloupnosti)

• zadávání hodnot aktuálních parametrů:

- při deklaraci příslušných instancí v sekci parametrů nadřazeného modulu: lze použít hodnot, výrazů i funkcí, např. pro generování náhodných čísel (viz manuál),
- během simulace:
  - pomocí konfiguračního souboru omnetpp.ini,
  - interaktivně z klávesnice,

Sekce parametrů modulu

Příklad: deklarace formálních parametrů modulu

```
 simple Node
       parameters:
         a, b, pocet:  numeric,  // formální parametry modulu Node
         c:            string; .
       gates:……;
endsimple
```

• přířazení aktuálních parametrů: v sekci submodulů

```
module  M                        // deklarace typu modulu M
    parameters:    // formální parametry modulu M
       p1, p2, d_simul:  numeric,
                         bb : bool;
   gates:……;
   submodules: ……;
      sm1: Node ;        // deklarace instance sm1 typu Node
                parameters:  // aktuální parametry pro sm1:
                a = p1+p2,
                b = bb == true ?  p1 : p2,
                c = “token“ ,
                // následuje interaktivní přiřazení parametru pocet
                pocet = input ( 50,  “celkovy pocet  ” );
     connections:…….;
endmodule
```

Přiřazení hodnot aktuálních parametrů:

možnosti přiřazení :

- ve struktuře při deklaraci vnitřní instance,
- na začátku simulace (viz soubor omnetpp. ini nebo interaktivně),
- na všech úrovních zadávání hodnot aktuálních parametrů lze použít výrazů, podporované operace:
  - aritmetické: *, /, %, +, -,
  - posuvy: <<, >>
  - logické paralelní (bitwise): & (and), | (or), # (xor), ~ (negace)
  - relační: ==, !=, >, >=, <, <=)
  - logické: (&& (and), || (or), ## (xor), !(negace))
  - podmíněný operátor (B) ? V1 : V2; (if (B) V1 else V2)
- ve výrazech lze volat běžné matematické funkce jazyka C: např. exp( ), sin ( ) , atd. a funkce pro generování pseudonáhodných čísel,
- hodnoty numerických parametrů (long, double) s časovým významem lze specifikovat pomocí časových jednotek (ns, us, ms, s, m, h, d),

  příklad: **parameters**: delay = 70 ns;

- není-li formální parametr specifikován jako **const**, je příslušný výraz vyhodnocován při každém volání parametru,

Sekce vstupů a výstupů modulu

účel: definovat rozhraní modulu pomocí jednosměrných portů,

(s výjimkou přímo zasílaných zpráv jsou zprávy posílány a přijímány výhradně prostřednictvím portů)

- syntax:

gates: in: < identifikátor vstupního portu > |

< identifikátor vstupního portu > [ ]

out: < identifikátor výstupního portu > |

< identifikátor výstupního portu >[ ] ;

- nespecifikované pole vstupních nebo výstupních portů má rozměr 0,

Příklad: deklarace portů

```
gates:   in :  vst1,  vst2,  vst3;        // 3 jednoduché vstupní porty
         out: o1,  o2 [  ] ;
         // o1..jednoduchý výstupní port
         // o2.. pole s nespecifikovaným počtem výstupních portů,
         out: o3 [2];                     //  toto nelze
```

Dodatečná specifikace počtu portů:

– v deklaraci konkrétní instance daného typu v sekci **gatesises**,

– velikost nespecifivaného pole portů lze dynamicky zvětšovat v sekci **connections** pomocí operátoru ++.

Sekce vstupů a výstupů modulu

Příklad: určení dimenze rozhraní v instancích typů s nespecifikovaným počtem portů

• předpoklad: existuje typ Node: **simple** Node

parameters: …….;

gates:

in : vst [ ] ;

out : vyst [ ] ;

endsimple

• specifikace rozhraní konkrétních instancí typu Node:

module M // deklarace typu modulu M

parameters: pocet;

submodules

sm1: Node ; // instance sm1 typu Node

parameters: ….; // aktuální parametry sm1

gatesizes: vst [2] , vyst [3] ; // počty portů sm1

sm2: Node ; // instance sm2 typu Node

parameters:……; // aktuální parametry sm2

// následuje specifikace počtu portů instance sm2 na // základě parametru instance modulu M

gatesizes: vst [pocet] , vyst [počet + 1] ;

endmodule

Parametrizace struktury složeného modulu

Přiřazování aktuálních parametrů umožňuje:

- použití výrazů a matematických funkcí C++ : viz manuál,
- použití funkcí pro generování náhodných hodnot (pokud nejde o typ const, pak přiřazení při každém čtení),

Příklad : parametrizace počtu dílčích instancí složeného modulu

```
module M
  parameters: pocet : const, …. ;  // výčet form. parametrů M
  submodules:
    sbm1:  Node1 ;          // deklarace jedné instance typu Node1
           parameters: ……;  // aktuální parametry pro sbm1
    sbm2:  Node2 [ 2 ] ;    // deklarace dvou instancí typu Node2
           parameters: ……;
    sbm3:  Node3 [ pocet ];   // parametrizovaný počet instancí
           parameters: ……;
    sbm4:  Node3 [ 2 * pocet ];  // parametr = konstantní výraz
           parameters: ……;
     ……; // propojení instancí chybí
endmodule
```

Poznámky:

- hodnoty aktuálních parametrů instancí vektoru lze specifikovat podmíněně v sekci **parameters** (viz dále),
- počet portů v rozhraní konkrétních instancí v sekci **gatesizes** lze rovněž specifikovat podmíněně (viz dále),

Vytvoření výsledného modelu

- vytvoření sítě určitého typu (složený modul); propojení jeho dílčích instancí je dáno strukturou simulované sítě,
- syntax:

network <identifikátor sítě> : < typ složeného modulu >

parameters:……;

endnetwork

Poznámka: < typ složeného modulu > reprezentuje modul bez vstupů a výstupů

Příklad: výsledná specifikace simulačního modelu

- předpoklad: existuje typ složeného modulu M, jehož vnitřní struktura je vytvořena dle struktury simulované sítě,
- konkrétní model (network definition): deklarace sítě s jedinou instancí typu M:

network m : M // dříve definovaný typ

parameters:……; // stanovení aktuálních parametrů

// sítě m

endnetwork

Vnitřní struktura složeného modulu

- specifikace: vzájemné propojení dílčích instancí nebo propojení dílčí instance se vstupním nebo výstupním portem daného modulu,
- nelze vzájemně propojit vstupní porty nebo výstupní porty, systém podporuje pouze jednosměrná propojení typu point to point; pomocí těchto pak uživatel musí realizovat dvousměrná propojení nebo spojení typu „one to many “ nebo „ many to one “,
- propojení instancí je implicitně kontrolováno
- syntax:

connections: (nebo **connections nocheck)**:

< identifikátor instance > . < identifikátor portu > --> resp. <--

[<parametry komunikační linky >] --> resp. <--

< identifikátor instance > . < identifikátor portu > , ….;

< parametry komunikační linky > :: =

< identifikátor kanálu > | <přímá specifikace linky >

< přímá specifikace linky > :: =

delay < zpoždění kanálu (propagation delay) [sec] > ;

error < pravdepodobnost chyby jednoho bitu (bit error rate) >;

datarate < přenosová kapacita (data rate) [bit / sec] > ;

Poznámka: implicitní hodnoty parametrů:

0 pro **delay** a **error** ,

∞ pro **datarate**.

Parametry komunikačních linek

Příklad: jednosměrné spojení instancí a1 a b2

```
// následuje spojení výstupniho portu vyst instance a1 se
// vstupním portem vst instance b2; zpoždění = 0,
// pravděpodobnost chyby = 0, linka je nekonečně rychlá
a1 . vyst -->  b2 . vst;
```

Příklad: obousměrné spojení instancí a1 a b2 s definovanými parametry komunikačních linek

```
connections:
a1 . vyst  -->  delay 0.002    error:  1e-8
                datarate 128000                --> b2 . vst ;
 a1 . vst  <--  delay 0.002    error 1e-8
                datarate 128000  ;              <--  b2 . vyst ;
```

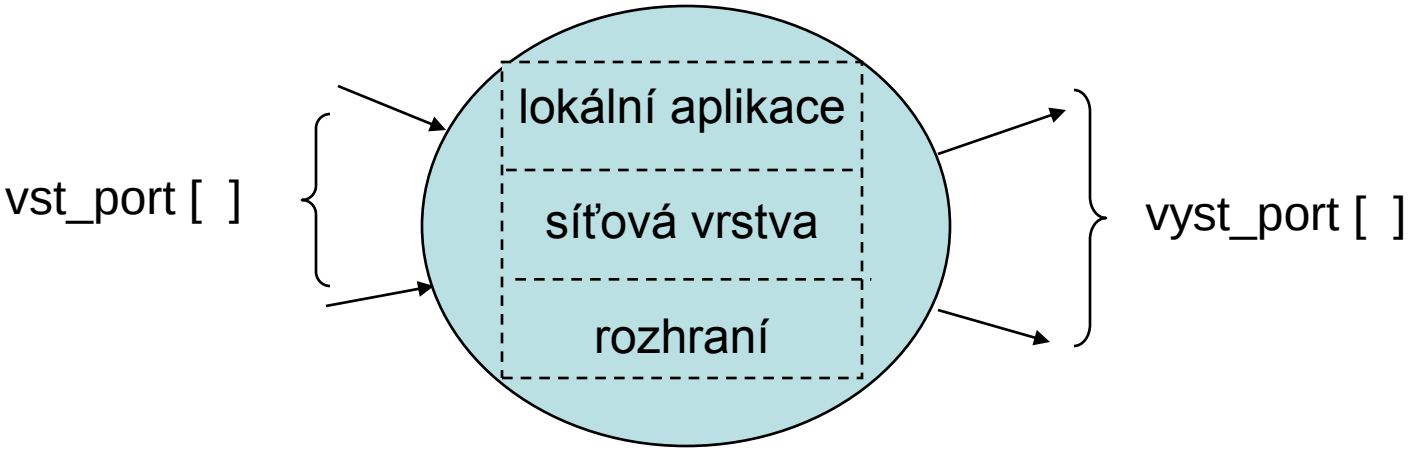
Poznámka: jiný ekvivalentní způsob deklarace obousměrného spojení se specifikovanými parametry:

```
channel   K1
          delay  0.002 ;         // sec
          error  1e-8  ;         // pravděpodobnost chyby bitu
          datarate 128000 ;  // bit / sec
endchannel
connections:    a1 . vyst  -->   K1   --> b2 . vst ;
                a1 . vst   <--   K1  <-- b2 . vyst ;
```

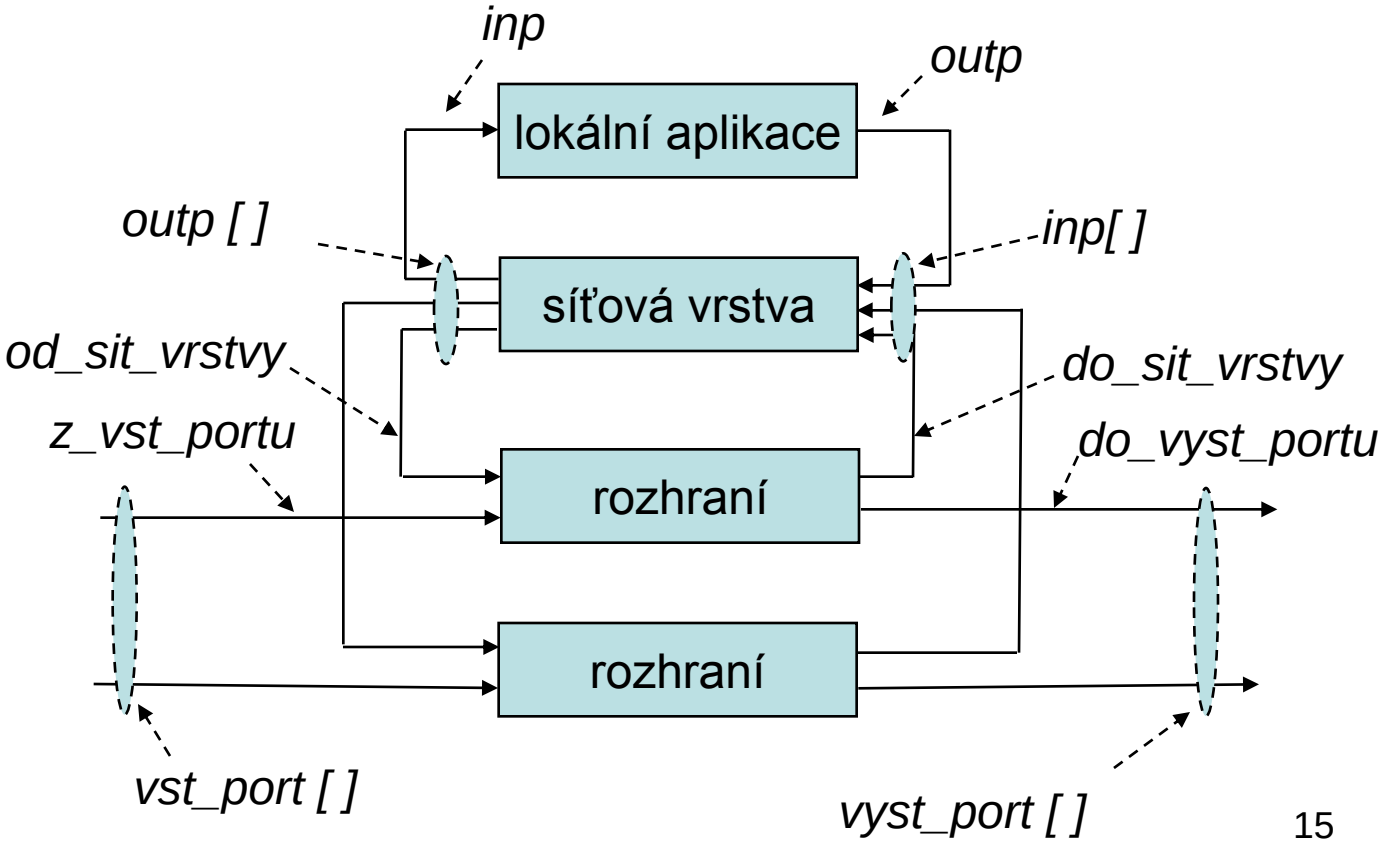
Příklad (vnitřní struktura směrovače)

Příklad: implementace vnitřní struktury směrovače.

• třívrstvá architektura



• propojení vrstev:



Příklad (vnitřní struktura směrovače)

• předpoklad: existence typů modulů:

```
simple Lok_aplikace
        parameters: ……..;
        gates:
            in: inp;
            out: outp;
endsimple
simple  Sit_vrstva
        parameters: ……..;
         gates:
             in:    inp   [ ];  // neurčený vektor vstupních portů
            out: outp  [ ];  // neurčený vektor výstupních portů
endsimple
simple  Rozhrani
        parameters: ……..;
        gates:
            in: od_sit_vrstvy, z_vst_portu;
           out: do_sit_vrstvy, do_vyst_portu;
endsimple
```

Poznámka: dimenzi neurčeného vektoru portů lze určit při deklaraci instance příslušného modulu pomocí parametru nebo odvodit z dimenze rozhraní nadřazeného modulu funkcí **sizeof** (<jméno portu>)

Příklad (vnitřní struktura směrovače)

```
module Smerovac
 parameters: pocet_portu ;     // parametry smerovace
 gates: in : vst_port [ ];  // neurčený vektor portů
          out: vyst_port [ ];
 submodules:
         la:     Lok_aplikace;            // zde bez parametrů
         sv:    Sit_vrstva;
                 parameters: ……;    // parametry sitove vrstvy
                 gatesizes:
                     inp   [ pocet_portu + 1],  // dimenze rozhraní
                     outp [ pocet_portu + 1];
                     // inp   [ sizeof ( vst_port )+1], //dimenze jinak
                     // outp [ sizeof ( vyst_port )+1];
        interf: Rozhrani [ pocet_portu ];    // počet dílčích instancí
 connections:                    // sekce spojení dílčích instancí
  for i = 0 .. pocet_portu –1 do // propojení sv,  interf[0] a interf[1]
           sv . outp [ i ] → interf [ i ] . od_sit_vrstvy;
           sv . inp   [ i ] ← interf [ i ] . do_sit_vrstvy;
           interf [ i ] . do_vyst_portu   → vyst_port [ i ] ;
           interf [ i ] . z_vst_portu      ←  vst_port [ i ] ;
  endfor;
  sv . outp [ pocet_portu ]  → la . inp;    // propojení  la  a  sv
  sv . inp   [ pocet_portu ] ←  la . outp;
endmodule
```

Příklad (vytvoření sítě)

```
simple Stanice                  // pro vnější spojení se směrovačem
gates:   in: in;   out: out;
endsimple

module Sit                      // typ sítě:  uzavřený systém !!!!!!!
   parameters: pocet_stanic;     // celkový počet stanic
   submodules:
       sm: Smerovac;
            parameters:pocet_portu = pocet _stanic;
            gatesizes: vst_port [ pocet _stanic ],
                       vyst_port [ pocet _stanic ];
       st:   Stanice [ pocet _stanic ];
   connections:
      for i = 0 .. pocet _stanic -1 do
            st [ i ] . out  --> sm . vst_port [ i ];
            st [ i ] . in  <-- sm . vyst_port [ i ];      endfor;
endmodule

network sit: Sit                    // výsledná instance sítě
parameters: pocet _stanic = 5;
endnetwork
```

Poznámka:

connections nocheck: potlačení kontroly propojení

Příklad: hyperkrychle

- specifikace stavebního modulu:

simple Node

gates: in : vst [ ]; **out**: vyst [ ] ;

endsimple

- specifikace vnitřní struktury hyperkrychle**:**

module Hypercube

parameters: rozmer: **numeric**;

submodules:

node: Node [2 ^ rozmer] ;

gatesizes: vst [rozmer],

vyst [rozmer] ;

connections:

for i = 0 . . 2^ rozmer - 1,

j = 0 . . rozmer - 1 **do**

node [i] . vyst [j] → node [i # 2^ j] . vst [j] ;

endfor;

endmodule

Operace:

^… mocnina,

#....XOR,

##....bitwise XOR, další viz manuál,

Struktura třírozměrné hyperkrychle

• význam propojení vnitřní struktury (uzly vyjádřeny binárně):

node [000] . vyst [0] → node [001] . vst [0] ;
node [000] . vyst [1] → node [010] . vst [1] ;
node [000] . vyst [2] → node [100] . vst [2] ;
node [001] . vyst [0] → node [000] . vst [0] ;
node [001] . vyst [1] → node [011] . vst [1] ;
node [001] . vyst [2] → node [101] . vst [2] ;
node [010] . vyst [0] → node [011] . vst [0] ;
node [010] . vyst [1] → node [000] . vst [1] ;
node [010] . vyst [2] → node [110] . vst [2] ;
node [011] . vyst [0] → node [010] . vst [0] ;
node [011] . vyst [1] → node [001] . vst [1] ;
node [011] . vyst [2] → node [111] . vst [2] ;
node [100] . vyst [0] → node [101] . vst [0] ;
node [100] . vyst [1] → node [110] . vst [1] ;
node [100] . vyst [2] → node [000] . vst [2] ;
node [101] . vyst [0] → node [100] . vst [0] ;
node [101] . vyst [1] → node [111] . vst [1] ;
node [101] . vyst [2] → node [001] . vst [2] ;
node [110] . vyst [0] → node [111] . vst [0] ;
node [110] . vyst [1] → node [100] . vst [1] ;
node [110] . vyst [2] → node [010] . vst [2] ;
node [111] . vyst [0] → node [110] . vst [0] ;
node [111] . vyst [1] → node [101] . vst [1] ;
node [111] . vyst [2] → node [011] . vst [2] ;

Operátor ++

- v případě nespecifikovaného pole portů nějaké dílčí instance lze v sekci **connections** dynamicky zvětšovat dimenzi tohoto pole pomocí operátoru ++:

Příklad: jiná forma zápisu vnitřní struktury modulu Směrovač

```
module Smerovac                 // bez parametru pocet_portu
  gates:    in : vst_port [ ];
            out: vyst_port  [ ];
  submodules:
              la:     Lok_aplikace;
              sv:    Sit_vrstva;
                     // zde bez konkretizace dimenze rozhrani
             interf: Rozhrani [ sizeof ( vst_port ) ];    /
  connections
    for i = 0 .. sizeof ( vst_port ) – 1 do
          sv . outp ++    →  interf [ i ] . od_sit_vrstvy ;
           //  inkrementace pole outp instance sv
          sv . inp   ++    ←  interf [ i ] . do_sit_vrstvy ;
          interf [ i ] . do_vyst_portu   -- > vyst_port [ i ] ;
          vstup_port [ i ] → interf [ i ] . z_vst_portu [ i ] ;
    endfor;
          sv . outp      →  la . inp;
          sv . inp       ←  la . outp ;
endmodule
```

Operátor index

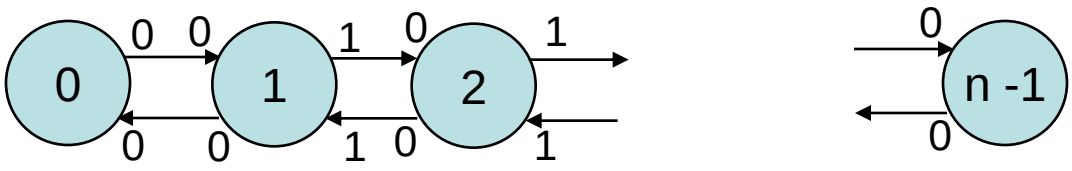
• ve specifikaci struktury lze použít operátor **index**, který určuje pořadí (počínaje 0) jednotlivých instancí určitého typu v poli instancí; toto pořadí může sloužit pro rozlišení instancí při jejich zapojování v nepravidelné struktuře či k odlišnostem v nastavení jejich parametrů

Příklad: nastavení individuálních aktuálních parametrů a individuálních dimenzí portů rozhraní jednotlivých instancí určitého typu

```
module …..;
  parameters:…….;   gates:……;
  submodules:
  sm: SMOD [ 8 ] ;   // deklarace vektoru instancí
      parameters if  index == 0:
      par1 = ……;     // hodnota par1 pro sm [ 0 ]
      ……..;
      parameters if  index == 7:
      par1 = ……;     // hodnota par1 pro sm [ 6 ]
      gatesizes if  index == 0:
      inpp [ .. ] ;      // dimenze vektoru portů pro sm [ 0 ]
      …….;
      gatesizes if  index == 7:
      outp [ .. ] ;      // dimenze vektoru portů pro sm [ 6 ]
  connections: ……;
endmodule
```

Příklad nepravidelné struktury

• propojení instancí téhož typu:



• předpoklad: existuje modul

simple M

gates: in : vst [ ]; // vektor s nespecifikovanou dimenzí

out: vyst [ ] ;

endsimple

module Sit

parameters: pocet : **numeric const**;

submodules:

m: M [pocet] ; // vektor instancí typu Node

gatesizes: vst [2], vyst [2] ;

// následuje změna dimenze pro krajní instance

gatesizes if index == 0 || index == n – 1:

vst [1], vyst [1] ;

connections:

for i = 0 . . pocet – 2 **do**

m [i] . vyst [i != 0 ? 1 : 0] → m [i + 1] . vst [0] ;

m [i] . vst [i != 0 ? 1 : 0] ← m [i + 1] . vyst [0] ;

endfor;

endmodule

Parametrizace typu modulu

NED umožňuje konkretizovat i typy dílčích modulů ve struktuře složeného modulu pomocí aktuálních parametrů,

- v jazyku NED je třeba definovat „formální typ modulu“, který bude při specifikaci struktury nadřazeného modulu zastupovat později určený aktuální typ; není však třeba definovat chování tohoto „zástupce“ v jazyku C++,
- zmíněný „formální typ modulu“ musí mít stejné vnitřní parametry i rozhraní jako aktuální moduly, které zastupuje

Příklad: struktura hyperkrychle, ve které potřebujeme dosazovat pro jednotlivé dimenze různé typy

```
// následuje specifikace „formálního typu modulu“:
simple Node
      gates:  in :  vst   [  ];  out: vyst [  ] ;
endsimple

// následuje specifikace vnitřní struktury nadřazeného modulu
module Hypercube
  parameters: typ_uzlu:   string,   // pro parametrizaci typu
                  rozmer:  numeric;
  submodules: node:  typ_uzlu [ 2 ^ rozmer ]  like Node;
   gatesizes:        // viz příklad  hyperkrychle
   connections: // viz příklad  hyperkrychle
endmodule
```

Pokračování příkladu

- předpoklad: existuje několik typů modulů s různým chováním ale se stejným rozhraním jako „formální modul Node“

```
simple U1
        gates:  in: vst [ ]; out vyst [ ] ;
endsimple
simple U2
        gates:  in: vst [ ]; out vyst [ ] ;
endsimple
simple U3
        gates:  in: vst [ ]; out vyst [ ] ;
endsimple
```

- formální parametr **typ_uzlu** (viz. modul Hypercube) umožní zvolit jeden z aktuálních typů: U1, U2, U3 pro instanci sit:

```
network sit: Hypercube
  parameter:
      rozmer = 5;
      // typ_uzlu =  “U3“ ;  pozor: toto zlobí a hlásí chybu
      // pomůže přesun  aktuálního parametru do souboru ……ini:
      sit . typ_uzlu  =  "U3" ;   // hierarchický přístup:  toto je OK
endnetwork
```