Road Traffic Simulation & OMNeT++

Martin Doležal

Road Traffic Simulation - požadavky

- Simulátor silniční dopravy
- Definice sítě
- Objekt křižovatka
- Objekt automobil
- Zdroj a cíl dopravních toků
- Simulace sítě jako SHO
- Výpočet statistických údajů
- Implementace v OMNeT++

Zdroj a cíl dopravních toků

- Zdroj a cíl dopravních toků – automobilů
- Výchozí statistické údaje
  - Počet průjezdů v daném směru
  - Toky mezi danými místy
  - Počet aut za hodinu
- Hierarchické uspořádání sítí

Automobil

- Aktivní automobil
  - Zná cíl cesty
  - Vlastní rozhodnutí, kudy pokračovat
  - Nese si informace o cestě
- Pasivní automobil
  - Zná cíl cesty
  - Síť samotná ho navádí nejlepší cestou
  - Neví, kudy pojede
  - Cesty v daném toku se průběžně mění

Křižovatka

- Má průběžné informace o celé síti
- Navádí automobily příslušnou cestou k cíli
- Přizpůsobuje se momentální situaci
  - Změna trasy – porucha, zácpa,...
  - Intervaly semaforů
- Na rozdíl od automobilu je to aktivní prvek

Silnice

- Spojuje navzájem
  - Křižovatky
  - Zdroje dopravních toků
  - Cíle dopravních toků
- 'Přenos automobilů' mezi křižovatkami
- Její parametry výrazně ovlivňují chování sítě
  - Délka silnice
  - Počet pruhů
  - Maximální rychlost
  - Pravděpodobnost poruchy - zablokování

Silniční síť

- Složená z definovaných objektů (křižovatka, silnice, zdroje a cíle toků)
- Momentální stav je určen stavy těchto objektů a zatížením sítě automobily

Příklady využití

- Simulace silniční sítě při její přestavbě
  - Omezení dopravy v daném úseku
  - Nové silnice či rozšíření stávajících
- Automatické navádění automobilů k cíli
  - Respektuje aktuální dopravní stav
  - Proměnné dopravní značení
  - Přizpůsobení délek intervalů u semaforů aktuální dopravní situaci

Souvislosti s počítačovou sítí

- Zřejmé souvislosti s počítačovou sítí
  - Automobily, křižovatky, silnice, zdroje toků dat,...
  - Packety, routery, kanály, servery,...

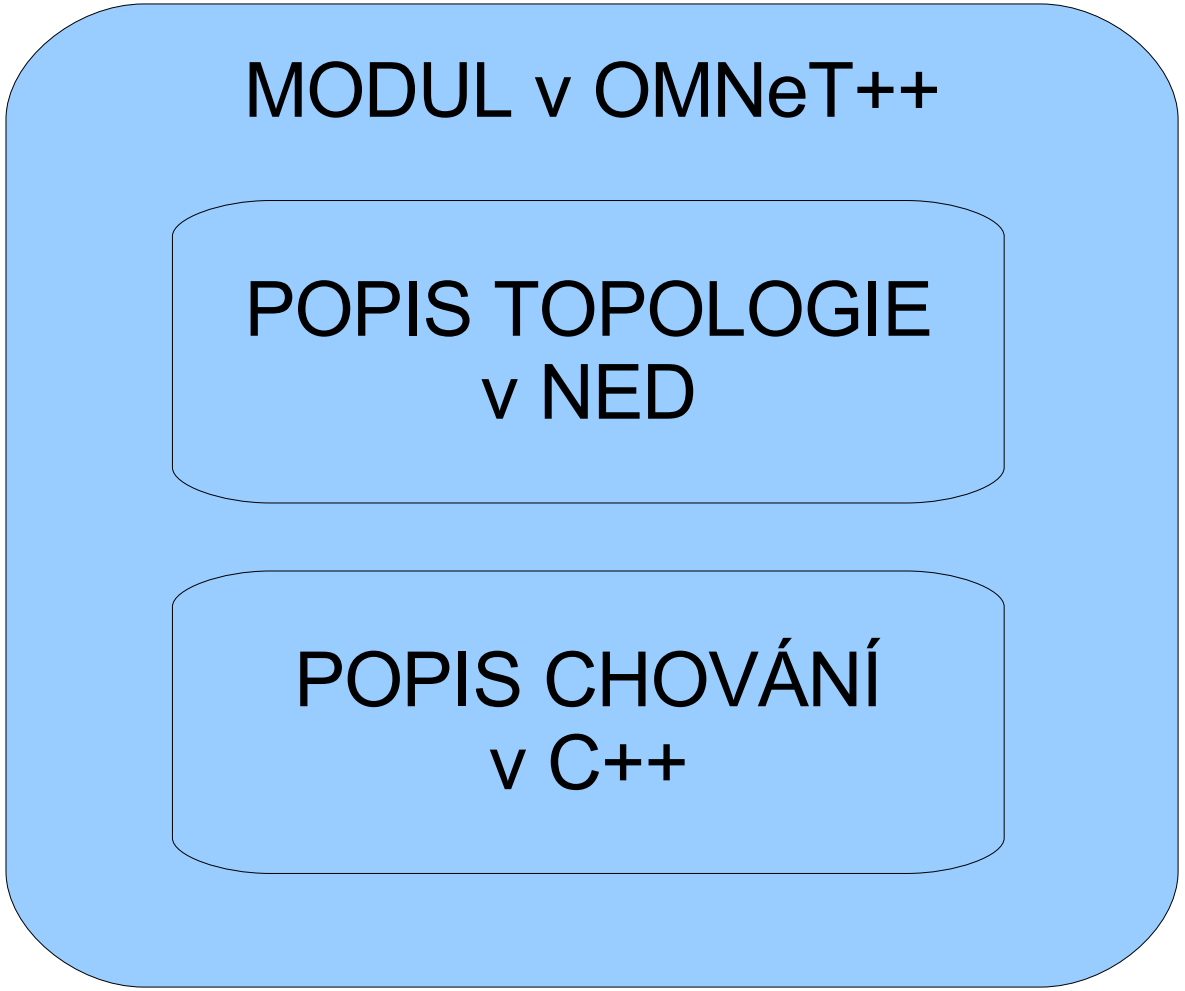
Diskrétní simulace

- Simulace systémů s diskrétními událostmi
- Dění v systému lze popsat pomocí událostí
- Mezi každými dvěma událostmi se „nic neděje“
- Událost trvá nulový čas
- Událostí je například
  - Počátek přenosu paketu
  - Konec přenosu paketu
  - Vypršení timeoutu

OMNeT++

- Objektově orientovaný modulární simulátor určený pro diskrétní simulace
  - Modelování telekomunikačních sítích
  - Modelování protokolů
  - Modelování distribuovaných systémů
  - Validace hardwarových architektur
  - Vyhodnocení výkonnosti SW systémů
  - A jiné systémy s diskrétními událostmi

OMNeT++



Zprávy

- Prostředek komunikace mezi moduly
- V simulátoru silniční dopravy
  - automobily
  - zpráva o nutnosti přepočítat směrovací tabulky
  - self-messages
- jsou zděděny od základní třídy `cMessage`

Self-messages

- Zpráva, kterou modul posílá sám sobě
- Typické využití jako časovač
  - první self-message se pošle při inicializaci
  - při přijmutí ji znovu naplánuji za další interval a provedu příslušnou akci
- **`scheduleAt`**`(time, msg)`
- **`cancelEvent`**`(msg)`

FES

Future Event Set – fronta událostí – zpráv

1. Zpráva, která má být doručena dříve, se zpracuje první. Jestliže jsou časy dvou zpráv stejné
2. Zpráva s větší prioritou (v OMNeTu jsou vyšší priority ty s menším číslem) se zpracuje první, jestliže jsou stejné
3. Zpráva dříve poslaná (naplánovaná) se zpracuje první

Simulační smyčka

```
initialize

while (FES not empty and
       simulation not yet complete)
{
  retrieve first event from FES
  t := timestamp of this event
  process event
}

finish simulation
```

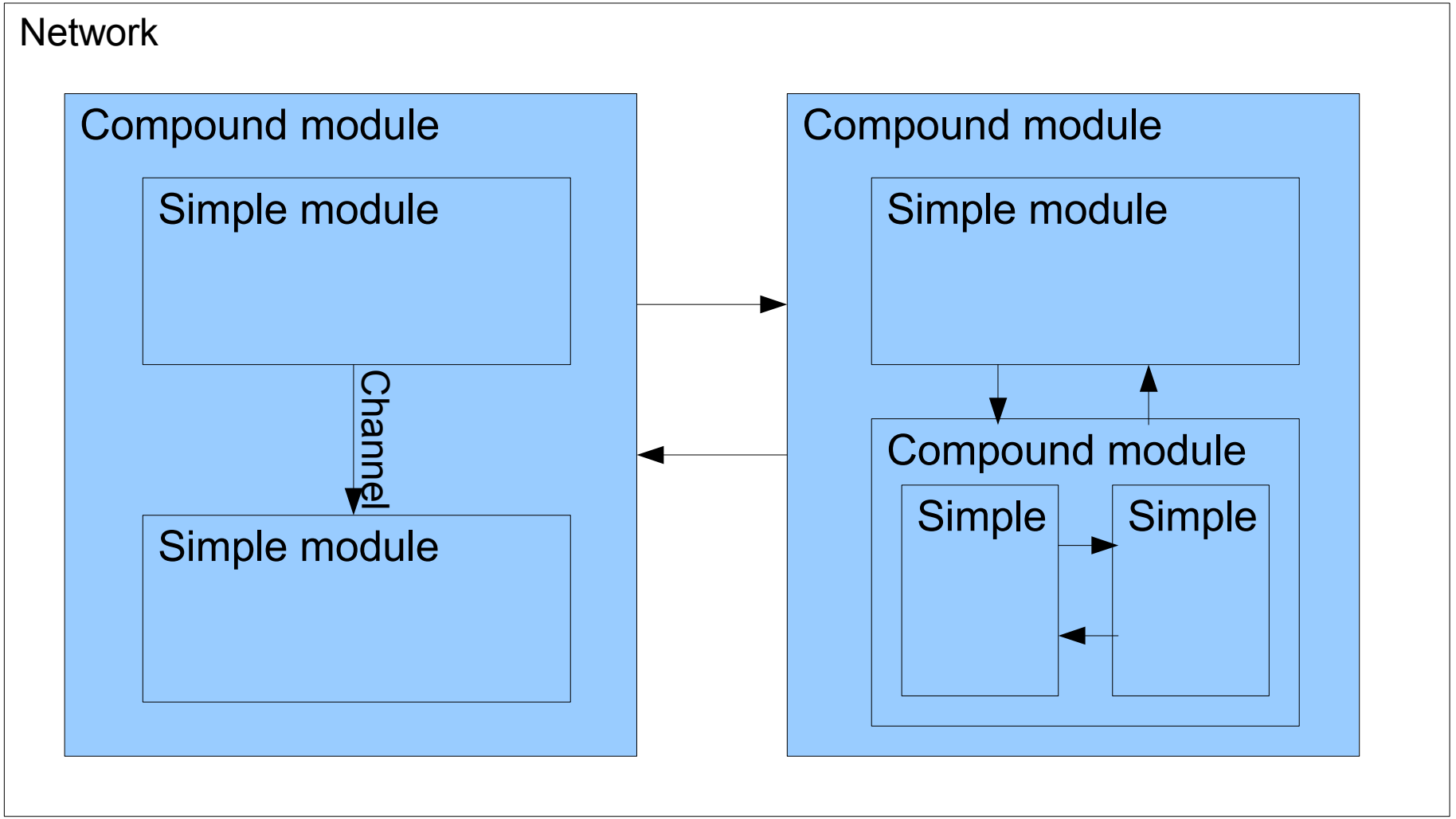
NED language

- Topologie sítě je popsána v jazyce NED
- NED umožňuje modulární popis sítě
- Popisy se ukládají do souborů s příponou .ned
- Simulátor načítá NED
  - Dynamicky při startu
  - nebo možný překlad do C++ a přímé přilinkování ke spouštěcímu souboru

NED language

- Popis v NEDu se skládá z následujících komponent:
  - Příkazy import
  - Definice kanálů
  - Definice simple a compound modulů
  - Definice sítě

NED language



NED language - import

Příkazem se vkládá externí soubor

```
import "ethernet"
```

NED language – simple module

```
simple SimpleModuleName
  parameters:
      //...
  gates:
      //...
endsimple
```

CarGenerator.ned

```
simple CarGenerator
    gates:
        out: out;
        in: in;
endsimple
```

crossroad.ned

```
simple CrossRoad
    parameters:
      throughDelay: numeric;
      type: numeric const;
    gates:
      out: outA, outB, outC, outD;
      in: inA, inB, inC, inD;
endsimple
```

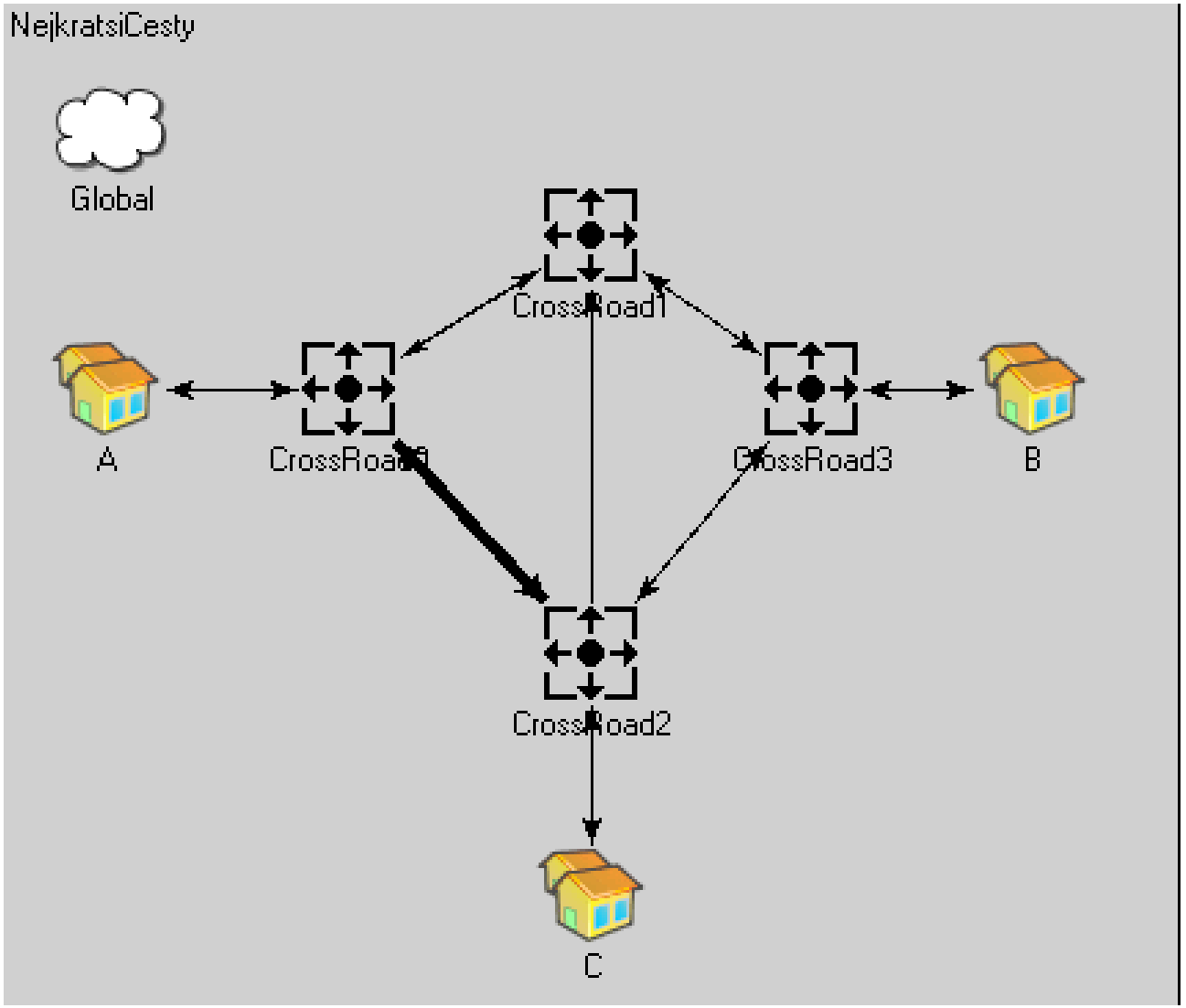
NED language – channel

```
channel MyChannel
  delay 0.0018
  error 1e-8
  datarate 128000
endchannel
```

NED language – compound module

```
module CompoundModule
  parameters:
      //...
  gates:
      //...
  submodules:
      //...
  connections:
      //...
endmodule
```

Příklad NED souboru s dopravní sítí



network2.ned

```
import
     "cargenerator",
     "crossroad",
     "global";
module NejkratsiCesty
    //...
endmodule

network network2 : NejkratsiCesty
endnetwork
```

module NejkratsiCesty

```
module NejkratsiCesty
    submodules:
        //...
    connections nocheck:
        //...
    display: "b=428,372";
endmodule
```

module NejkratsiCesty - submodules

```
submodules:
 Global: Global;
  display: "p=50,58;i=misc/cloud";
 CrossRoad0: CrossRoad;
  display: "p=136,152;i=old/bwgen";
 A: CarGenerator;
  display: "p=48,152;i=misc/town";
 //...
```

module NejkratsiCesty - connections

```
connections nocheck:
  A.out --> delay 3 --> CrossRoad0.inD;
  CrossRoad0.outD --> delay 10 --> A.in;
  //...
```

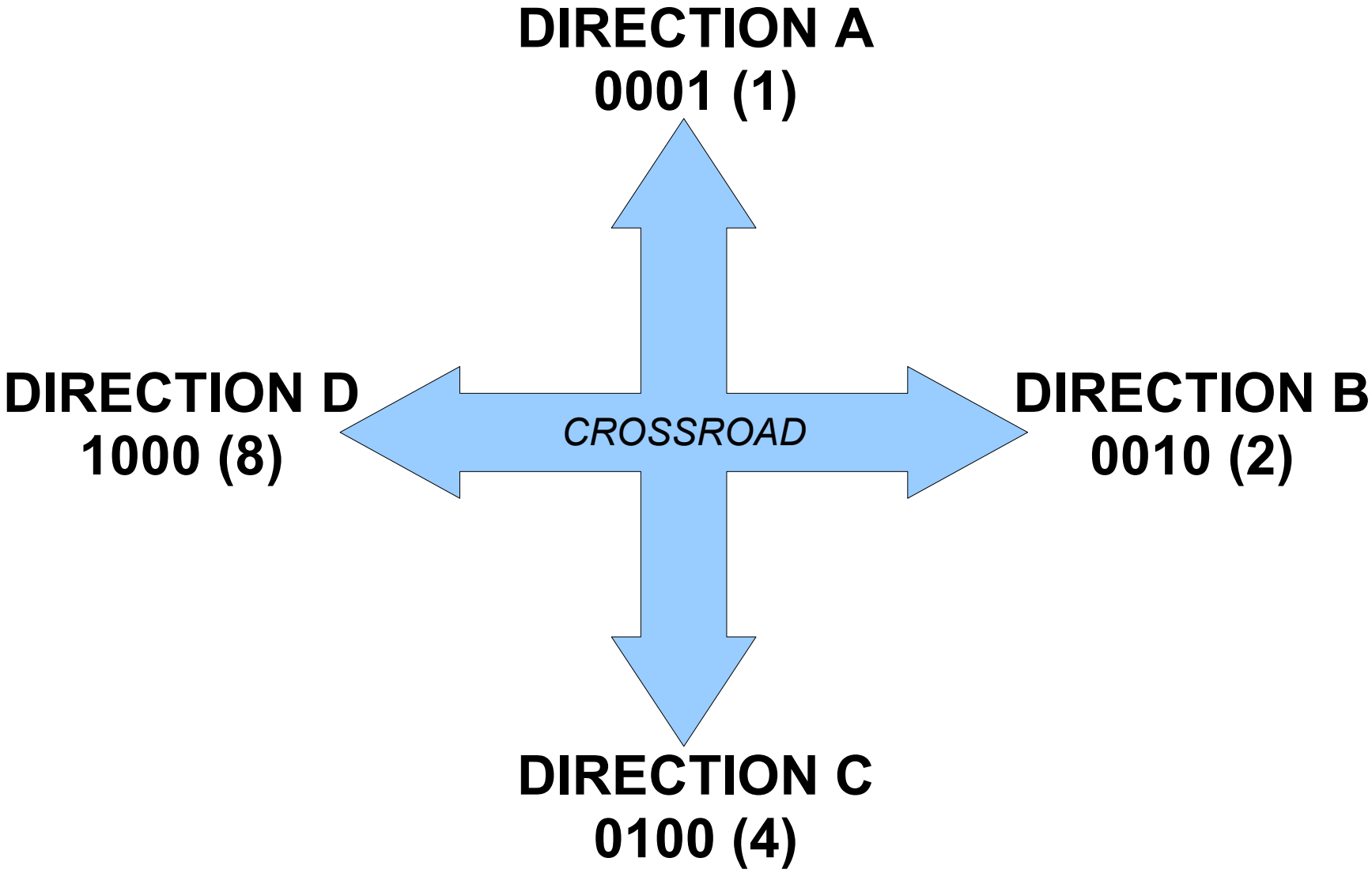
Ukázka

Tvorba NED souboru v GNED

Typy křižovatek

- Jsou definovány v externím souboru
- Pomocí jednoduché gramatiky je možno popsat chování křižovatky

Křižovatka


DIRECTION A
0001 (1)
DIRECTION D
1000 (8)
CROSSROAD
DIRECTION B
0010 (2)
DIRECTION C
0100 (4)

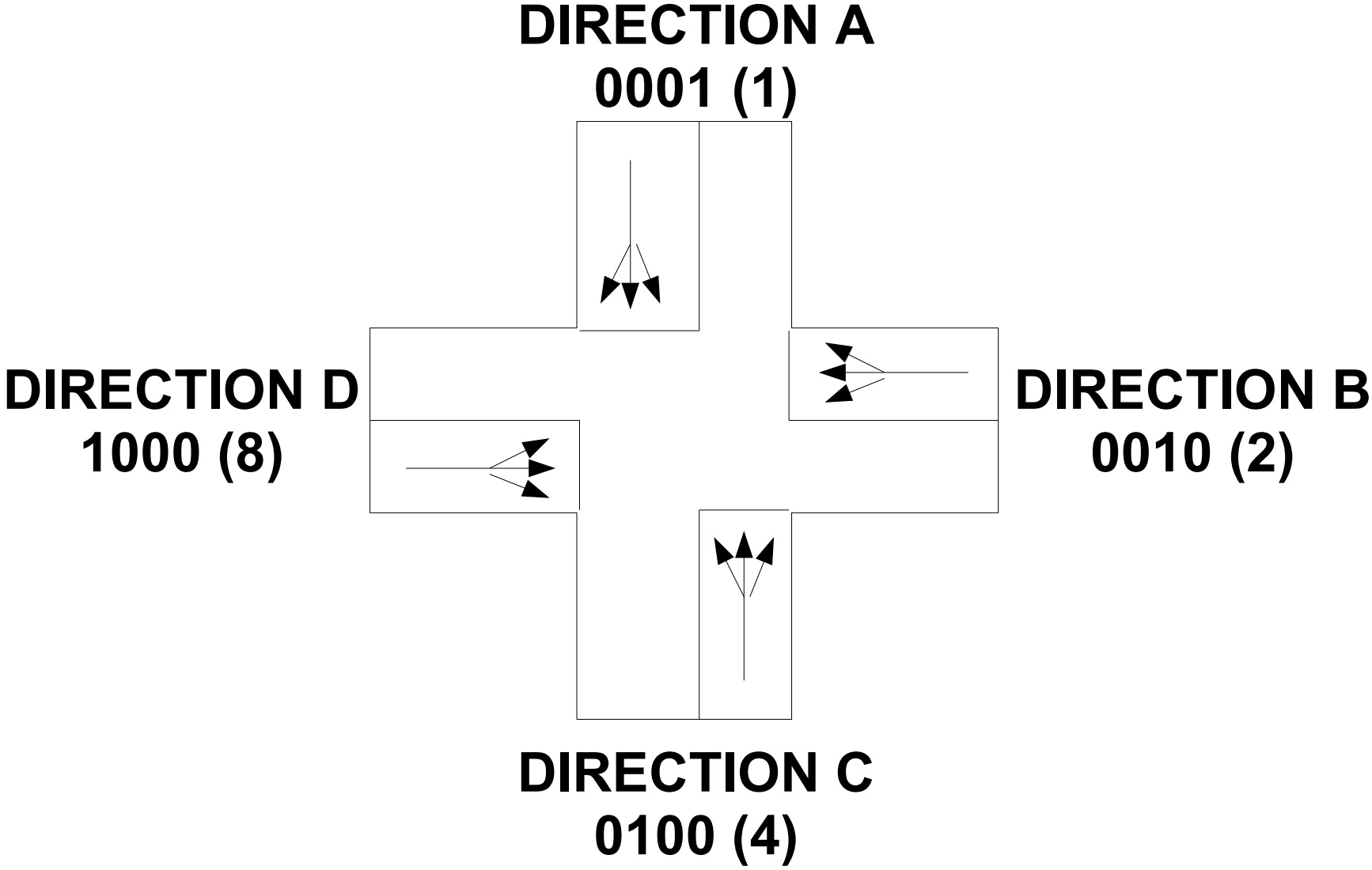
Bitové operace

- Směry jsou reprezentovány čísly
- Používají se bitové operace, například
  - Rotace doleva – získáme pruh nalevo od aktuálního
  - Auto jede do *autoCíl* a pruh vede do *pruhCíl*
    jedná se o vhodný pruh, pokud
    (autoCíl AND pruhCíl) != 0
  - apod.

Popis křižovatky

TYPE *Number TypeName*
LANES
(OneChar Chars)+
[
MAJOR *TwoChars*
]
[
INTERVALS
*Number(TwoChars)**
]

Příklad křižovatky



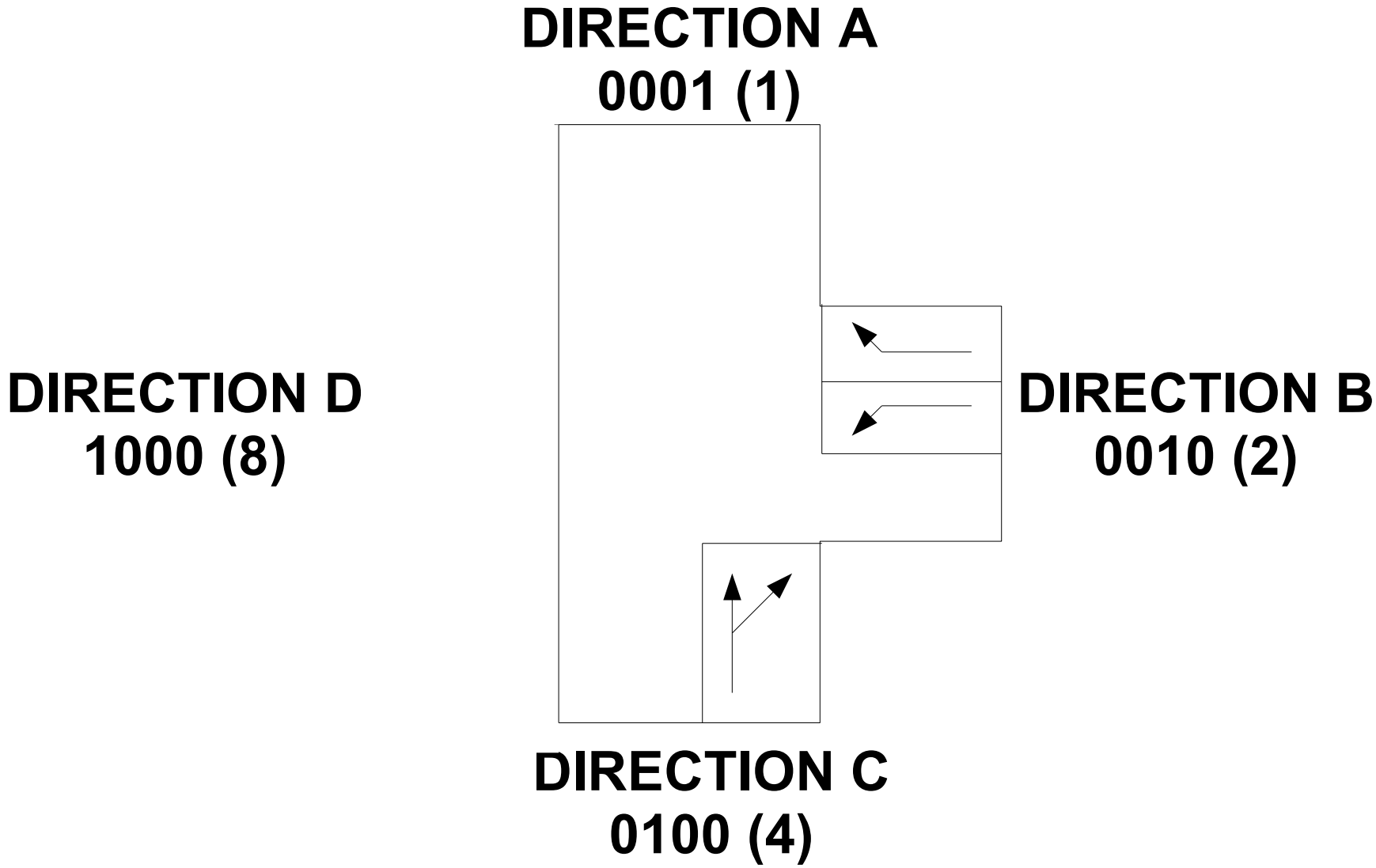
Popis předchozí křižovatky

```
TYPE 10 ExampleCrossRoad
LANES
A BCD
B ACD
C ABD
D ABC
MAJOR AC
```

Světelná křižovatka

```
TYPE 11 ExampleClockwise
LANES
A BCD
B ACD
C ABD
D ABC
MAJOR AC
INTERVALS
30 AB AC AD
30 BA BC BD
30 CA CB CD
30 DA DB DC
```

Příklad křižovatky 2



Popis předchozí křižovatky

TYPE *12 ExampleCrossRoad2*
LANES
B A C
C AB

Předdefinované typy

- Je možno použít některé předem definované typy křižovatek
  - Základní – 4 směry – přednost zprava
  - Kruhový objezd
- Možnost odkázat se na již definovaný typ a jenom změnit některé jeho vlastnosti (typicky intervaly semaforů)

Simple module v OMNet++

Třída zděděná od `cSimpleModule`, je nutno přetížit následující virtuální metody:

```
void initialize()

void handleMessage(cMessage *msg)

void activity()

void finish()
```

Constructor vs. initialize()

- Konstruktor – ukazatele na NULL
- initialize() – inicializace parametrů, proměnných, alokace dynamických dat, self-messages, apod.
- finish() – záznam statistik (žádné mazání)
- Destruktor – uvolnění paměti, zrušení self-messages, apod.

Varianta s activity()

- Podobné jako programování vlákna či procesu
- Kdekoliv v kódu je možno čekat na příchozí zprávu
- V kódu je možno na nějaký čas zastavit činnost modulu
- OMNeT++ tuto variantu příliš nedoporučuje

Varianta s activity()

```
activity()
{
    while (true)
    {
        // wait, receive,...
    }
}
```

Varianta s activity()

```
receive()
wait()
send()
scheduleAt()
cancelEvent()
end()
```

Varianta s handleMessage()

send()
scheduleAt()
cancelEvent()

~~**receive()**~~
~~**wait()**~~

CrossRoad::handleMessage()

```
void CrossRoad::handleMessage(cMessage *msg)
{
   if (msg->isSelfMessage()) {
      CRInfo *crinfo = check_and_cast<CRInfo*>(msg);
      switch (crinfo->type()) {
         case CRI_ONE_LANE_CHECK:
            carChecks(crinfo->lane());
            break;
         case CRI_ALL_LANES_CHECK:
            engagedPassageMsg[crinfo->from()]
                  [crinfo->to()] = NULL;
            allLanesCheck();
            break;
         case CRI_SEMAPHORE:
            changeSemaphore();
            break;
      }
      delete msg;
   }
```

CrossRoad::handleMessage() pokr.

```
else { // if it is not selfMessage
   Car* car = check_and_cast<Car*>(msg);
   carArrived(car, carArrivedToPos(car), carToPos(car));
}
```

CarGenerator::handleMessage()

```
void CarGenerator::handleMessage(cMessage *msg)
{
   if (!msg->isSelfMessage()) {
      // not selfMessage -> it is a car -> delete it
      ev << simTime() << " CarGenerator: deleting car";
      delete msg;
      return;
   }

   scheduleAt(simTime() + uniform(...), msg);

   car = new Car(messageCount-1);

   // send it to output
   send(car, "out");
}
```

Průjezd křižovatkou

- Je první ve frontě?
- Má zelenou?
- Má přednost?
- Má před sebou místo?
- Naplánuje možnost vyjetí následníka
- Naplánuje možnost vyjetí těch aut, která mu dávala přednost
- Přeplánování zprávy

omnetpp.ini

Konfigurační soubor

```
[General]
network = network1
sim-time-limit = 100h
cpu-time-limit= 300s
ini-warnings = yes
```

omnetpp.ini

```
[Run 1]
description="Hradec Kralove nadrazi"
**.throughDelay = 3.50;
**.Global.autoIntervalLength = true
**.Global.flows="flow1.txt"
network1.CEZ.type = 15
network1.Koruna.type = 11
**.type = 0
**.typesFile = "types.txt"
output-vector-file = output.vec
```

Ukázka

Road Traffic Simulation

OMNeT++

www.omnetpp.org

Road Traffic Simulation & OMNeT++

KONEC